Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Augustinas Jukna IFF-0/3

Studentas

Doc. Renata Burbaitė

Dėstytoja

Kaunas 2021

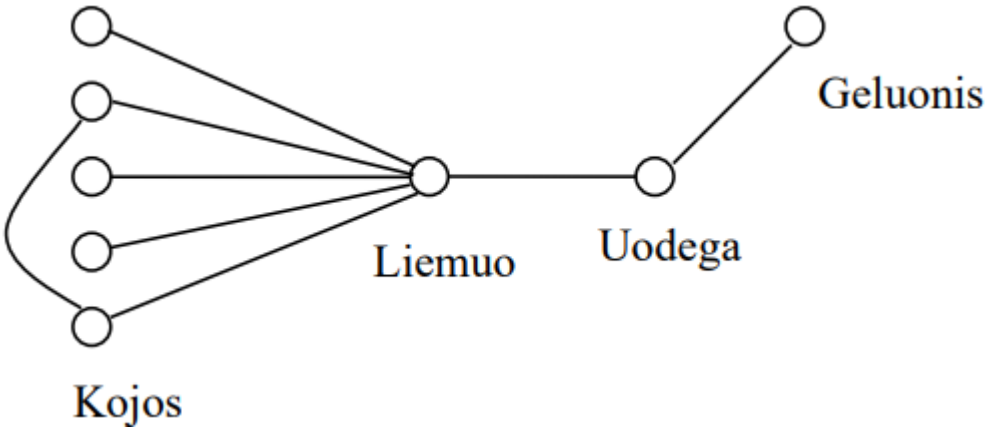
TURINYS

1. Rekursija (L1)

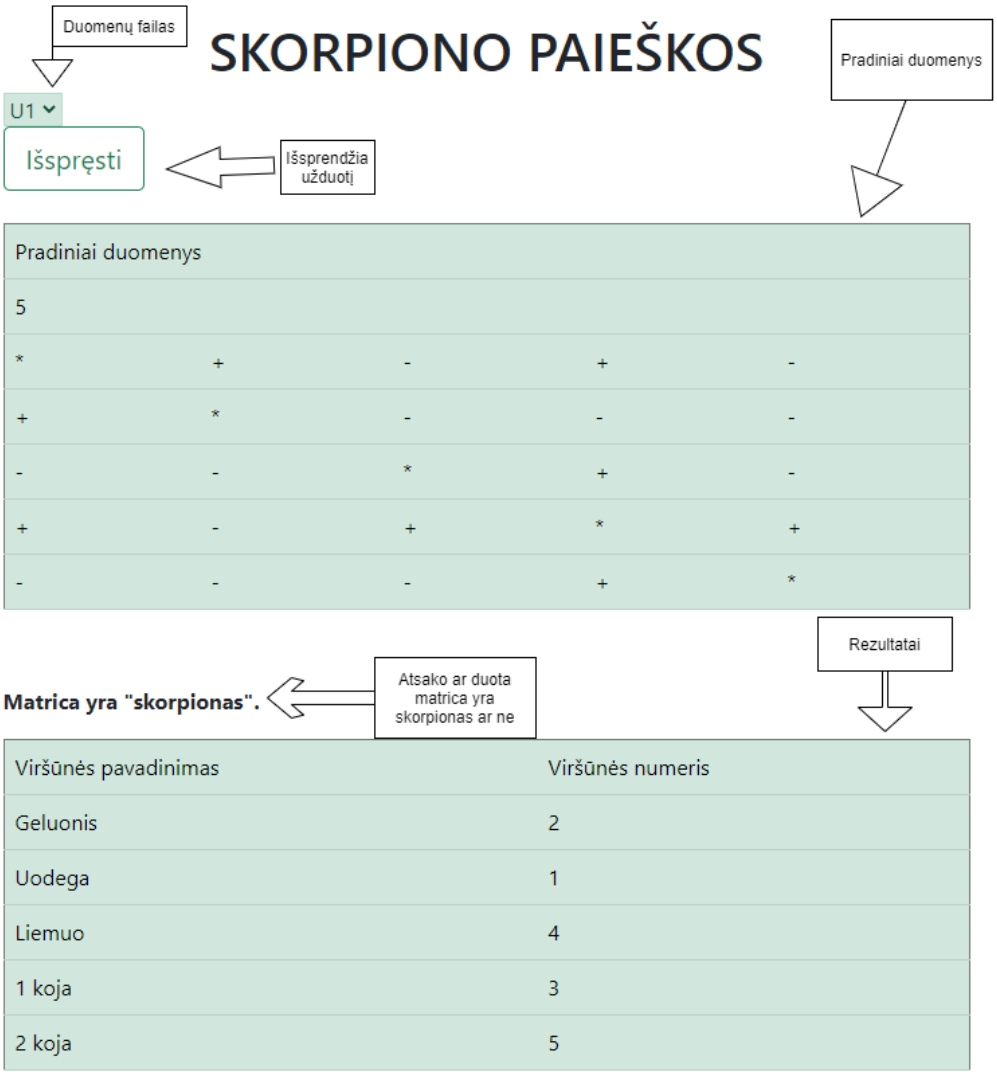
1.1. Darbo užduotis

LD_8. Skorpionas.

Grafas – tai viršūnių ir jas jungiančių briaunų visuma. Tarp dviejų viršūnių gali būti tik viena briauna. Brėžiniuose grafo viršūnės dažnai vaizduojamos mažais apskritimais, o briaunos – linijomis, jungiančiomis šiuos apskritimus. Grafas, turintis n viršūnių yra „skorpionas“, jei yra viena viršūnė (geluonis), sujungta viena briauna su kita viršūne (uodega). Uodega dar viena briauna turi būti sujungta su trečia viršūne (liemeniu). Viršūnė-liemuo jungiama su likusiomis viršūnėmis (kojomis) atskiromis briaunomis. Kai kurios viršūnės-kojos gali būti sujungtos tarpusavyje. Duomenys surašyti tekstiniame faile ’U3.txt’. Pirmoje failo eilutėje yra parašytas sveikasis skaičius n ($5 \leq n \leq 50$). n nurodo grafo viršūnių skaičių. Toliau eilutėmis, kurių kiekvienoje yra n simbolių, užrašyta grafo matrica V(n,n). V[i,j]=’-’, jei tarp i-osios ir j-osios viršūnių nėra briaunos ir V[i,j]=’+’, jei tarp i-osios ir j-osios viršūnių yra briauna. V[i,i]=’*’. Rezultatai. Išveskite pranešimą, ar įvestą matricą atitinkantis grafas yra „skorpionas“, ar ne. Jei taip, nurodyti, kuri viršūnė yra „geluonis“, kuri „uodega“, kuri „liemuo“, kurios viršūnės yra „kojos“.



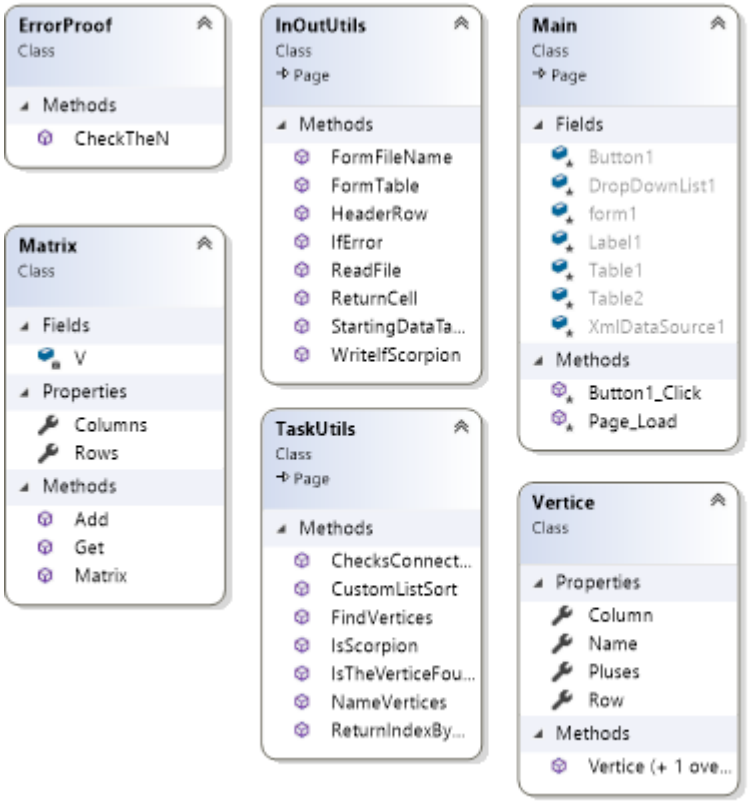
1.2. Grafinės vartotojo sąsajos schema



1.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
DropDownlist#DropDownList1	CssClass	alert-success
Button#Button1	Text	„Išspręsti“
Table#Table1	GridLines	Both
Table#Table1	CssClass	table table-success table-hover
Table#Table2	GridLines	Both
Table#Table2	CssClass	table table-success table-hover

1.4. Klasių diagrama



1.5. Programos vartotojo vadovas

Programa priima nuo 5 iki 50 imčių imtinai. Tereikia pasirinkti duomenų failą („U1“, „U2“, „U3“, „U4“) ir paspausti „Išspręsti“ mygtuką.

Jeigu norima sukurti ar redaguoti pradinius, duomenis, serverio „App_Data“ aplanke galima pasirinkti vieną iš jau esančių duomenų failų (pavyzdžiui „U1.txt“) ir jame keisti duomenis.

Pradinių duomenų sudarymo gidas:

1. Pirma eilutė dokumente – kiek stulpelių bei eilučių turės matrica. **Svarbu:** programa nepriims daugiau negu 50 ar mažiau negu 5 skaitmens.
2. Sekančiose eilutėse yra duomenų išdėstymas. Eilučių simbolių kiekis privalo būti lygus stulpelių kiekiui.
3. Yra galimos trys ženklų variacijos: ‘+‘ – jungtis tarp i-osios ir j-osios viršūnės, ‘-‘ – jungties nėra, ‘*‘ – viršūnės koordinatės.

   Pavyzdžiai: Matrica[1, 2] = ‘+‘ - tarp pirmos ir antros viršūnių yra jungtis.

   Matrica[2,2] = ‘*‘ – antros viršūnės koordinatės matricoje.

4. Tokia matrica laikoma skorpionu, kuri:
    a. Turi ne daugiau nei 50 ar ne mažiau nei 5 viršūnes;
    b. turi vieną geluonies viršunę, kuri jungiasi su uodega;
    c. uodegą, kuri jungiasi su geluonimi ir liemeniu;
    d. liemenį, kuris jungiasi su uodega bei viena ar daugiau kojų;
    e. kojos/koja, kuri/kurios jungiasi su liemeniu (gali jungtis ir tarpusavyje).

Žemiau pateiktoje lentelėje, bus pateikti pradiniai duomenys.

Toliau – rezultatas ar matrica yra skorpionas bei rezultatų lentelė.

1.6. Programos tekstas

Matrix.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab_1_WebApp
{
    public class Matrix
    {
        private char[,] V;
        public int Rows { get; private set; }
        public int Columns { get; private set; }

        public Matrix(int n)
        {
            this.V = new char[n, n];
            this.Rows = n;
            this.Columns = n;
        }
        /// <summary>
        /// Adds a char to a specific place in the container
        /// </summary>
        /// <param name="i">row to put the object in</param>
        /// <param name="j">column to put the object in</param>
        /// <param name="character">object</param>
        public void Add(int i, int j, char character)
        {
            this.V[i, j] = character;
        }

        /// <summary>
        /// Gets an object from a specific place
        /// </summary>
        /// <param name="i">row to take object from</param>
        /// <param name="j">column to take object from</param>
        /// <returns>the object</returns>
        public char Get(int i, int j)
        {

            return this.V[i, j];
        }



    }
```

}

InOutUtils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Lab_1_WebApp
{
    public class InOutUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Reads data file
        /// </summary>
        /// <param name="AllLines">Array of all the data</param>
        /// <returns>returns a Matrix class object</returns>
        public static Matrix ReadFile(string[] AllLines)
        {
            int n = Int32.Parse(AllLines[0]);
            if (!ErrorProof.CheckTheN(n))
            {
                return null;
            }

            Matrix allData = new Matrix(n);
            for (int i = 0; i < allData.Rows; i++)
            {
                for (int j = 0; j < allData.Columns; j++)
                {
                    char c = AllLines[i + 1][j];
                    allData.Add(i, j, c);
                }
            }
            return allData;
        }

        /// <summary>
        /// Forms a file name to fit system's settings
        /// </summary>
        /// <param name="dropDownList">data file input</param>
        /// <returns>returns a fileName</returns>
        public static string FormFileName(DropDownList dropDownList)
        {
            string fileName = "App_Data/" + dropDownList.SelectedValue + ".txt";
            return fileName;
        }

        /// <summary>
        /// Changes label's text based on if a given matrix is a scorpion or not
        /// </summary>
        /// <param name="label">object to change text</param>
        public static void WriteIfScorpion(Label label)
        {
            label.Text = @"<strong>Matrica yra ""skorpionas"".</strong>";
        }
        /// <summary>
        /// If the matrix is not a scorpion, then changes label's text to fit
accordingly
        /// </summary>
```

```
/// <param name="label">object to change the text</param>
public static void IfError(Label label)
{
    label.Text = @"<strong>Ši matrica nėra ""skorpionas"".</strong>";
}
/// <summary>
/// Returns a made cell
/// </summary>
/// <param name="text">cell's text input</param>
/// <returns>a made cell</returns>
public static TableCell ReturnCell(string text)
{
    TableCell cell = new TableCell();
    cell.Text = text;
    return cell;
}
/// <summary>
/// Creates table's header row
/// </summary>
/// <returns>a made header row for table</returns>
public static TableRow HeaderRow()
{
    TableRow row = new TableRow();
    row.Cells.Add(ReturnCell("Viršūnės pavadinimas"));
    row.Cells.Add(ReturnCell("Viršūnės numeris"));
    return row;
}
/// <summary>
/// Forms a full table from inputs
/// </summary>
/// <param name="table">Displayed table</param>
/// <param name="vertices">List of all the vertices</param>
public static void FormTable(Table table, List<Vertice> vertices)
{
    table.Rows.Add(HeaderRow());
    for (int i = 0; i < vertices.Count; i++)
    {
        TableRow row = new TableRow();
        row.Cells.Add(ReturnCell(vertices[i].Name));
        row.Cells.Add(ReturnCell((vertices[i].Row + 1).ToString()));
        table.Rows.Add(row);
    }
}

/// <summary>
/// Fills a table row with empty cells
/// </summary>
/// <param name="row">row to be filled</param>
/// <param name="columns">how many cells to add</param>
public static void FillTableRow(TableRow row, int columns)
{
    for (int i = 0; i < columns; i++)
    {
        row.Cells.Add(ReturnCell(""));
    }
}

/// <summary>
/// Creates starting data table for comparison
/// </summary>
/// <param name="table">table to display</param>
/// <param name="matrix">data container</param>
public static void StartingDataTable(Table table, Matrix matrix)
{
    TableRow row0 = new TableRow();
```

```
            TableCell cell = ReturnCell("Pradiniai duomenys");
            cell.ColumnSpan = matrix.Columns;
            row0.Cells.Add(cell);

            TableRow row1 = new TableRow();
            row1.Cells.Add(ReturnCell(matrix.Rows.ToString()));
            FillTableRow(row1, matrix.Columns - 1);

            table.Rows.Add(row0);
            table.Rows.Add(row1);
            for (int i = 0; i < matrix.Rows; i++)
            {
                TableRow rowTemp = new TableRow();
                for (int j = 0; j < matrix.Columns; j++)
                {
                    rowTemp.Cells.Add(ReturnCell((matrix.Get(i, j)).ToString()));
                }
                table.Rows.Add(rowTemp);
            }
        }

        /// <summary>
        /// Writes lines from vertice's list
        /// </summary>
        /// <param name="allLines">array of all the lines to write</param>
        /// <param name="index">index of line to start writing to</param>
        /// <param name="vertices">list of all the vertices</param>
        public static void WriteLines(string[] allLines, int index, List<Vertice>
vertices)
        {
            for (int i = 0; i < vertices.Count; i++)
            {
                allLines[index] = String.Format("Viršūnė: {0, -10} | Numeris:
{1}", vertices[i].Name, vertices[i].Row + 1);
                index++;
            }
        }

        /// <summary>
        /// Creates a string array to hold all the lines
        /// </summary>
        /// <param name="matrix">data matrix</param>
        /// <param name="vertices">all the vertices list</param>
        /// <returns>returns a made string array</returns>
        public static string[] WriteData(Matrix matrix, List<Vertice> vertices)
        {
            string[] AllLines = new string[matrix.Rows + vertices.Count + 2];
            AllLines[0] = "Pradiniai duomenys";
            AllLines[1] = String.Format("n = {0}", matrix.Rows);
            for (int i = 2; i <= matrix.Rows + 1; i++)
            {
                string line = "";
                for (int j = 0; j < matrix.Columns; j++)
                {
                    line += matrix.Get(i - 2, j);
                }
                AllLines[i] = line;
            }
            return AllLines;
        }

    }
}
```

Vertice.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab_1_WebApp
{
    public class Vertice
    {
        public string Name { get; set; }
        public int Row { get; set; }
        public int Column { get; set; }
        public int Pluses { get; set; }

        public Vertice(int row, int column)
        {
            this.Row = row;
            this.Column = column;
        }
        //Empty constructor
        public Vertice()
        {
            this.Row = -1;
            this.Column = -1;
        }
    }
}
```

TaskUtils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Lab_1_WebApp
{
    public class TaskUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Finds all the vertices and lists them (uses recursion)
        /// </summary>
        /// <param name="matrix">all data container</param>
        /// <param name="startRow">starting row</param>
        /// <param name="vertices">list of all the vertices</param>
        public static void FindVertices(Matrix matrix, int startRow, List<Vertice>
vertices)
        {
            int plusesCount = 0;
            Vertice vertice = new Vertice();
            for (int j = 0; j < matrix.Columns; j++)
            {
                if (matrix.Get(startRow, j) == '*') //if char == '*', that means,
it's i and j, will be the row and the column of the vertice
                {
                    vertice.Row = startRow;
                    vertice.Column = j;
                }
```

```
                else if (matrix.Get(startRow, j) == '+') //counts how many
connections does this vertice has
                {
                    plusesCount++;
                }
            }
            vertice.Pluses = plusesCount;
            vertices.Add(vertice);

            if (startRow == matrix.Rows - 1) //returns to prevent errors
            {
                return;
            }

            else
            {
                FindVertices(matrix, startRow + 1, vertices);
            }
        }

        /// <summary>
        /// Sorts through all the vertices and finds their hierarchy
        /// </summary>
        /// <param name="matrix">data container</param>
        /// <param name="vertices">list of all the vertices</param>
        public static void NameVertices(Matrix matrix, List<Vertice> vertices)
        {
            string sting = "Geluonis", tail = "Uodega", waist = "Liemuo", leg =
"koja";
            bool flag1 = true, flag2 = true; //keeps method usage in check, so
that certain methods would be used once only
            int legCount = 0;
            for (int i = 0; i < vertices.Count; i++)
            {
                Vertice vertice = vertices[i];
                if (vertice.Pluses == 1 && flag1)
                {
                    for (int j = 0; j < vertices.Count; j++)//starts a new loop to
find a suitable vertice that connects to it
                    {
                        Vertice vertice2 = vertices[j];
                        if (ChecksConnection(matrix, vertice.Row, vertice2.Row) &&
vertice2.Pluses == 2)
                        {
                            vertices[j].Name = tail;
                            vertices[i].Name = sting;
                            i = 0; //starts a new cycle of loop to not miss any
vertices
                            flag1 = false;  //to keep the method from repeating
                            break;
                        }
                    }
                }

                if (IsTheVerticeFound(vertices, tail) && vertice.Pluses >= 2 &&
flag2)
                {
                    Vertice vertice2 = vertices[ReturnIndexByName(vertices,
tail)];
                    if (ChecksConnection(matrix, vertice.Row,
vertice2.Row))//checks if  both of the vertices have a connection ('+')
                    {
                        vertices[i].Name = waist;
                        i = 0;
```

```
                        flag2 = false; //to keep the method from repeating
                    }
                }

                if (IsTheVerticeFound(vertices, waist) && vertice.Pluses >= 1 &&
!flag1 && !flag2) //this method will start the last, because both flag1 and flag2
have to be false
                {
                    Vertice vertice2 = vertices[ReturnIndexByName(vertices,
waist)];
                    if (ChecksConnection(matrix, vertice.Row,
vertice2.Row))//checks connection
                    {
                        legCount++;//counts the legs
                        vertices[i].Name = legCount + " " + leg;
                    }
                }

            }
        }

        /// <summary>
        /// Checks if the vertices connect together
        /// </summary>
        /// <param name="matrix">data container</param>
        /// <param name="vertice1Row">first vertice to check</param>
        /// <param name="vertice2Row">second vertice to check</param>
        /// <returns>returns a true or false statement</returns>
        public static bool ChecksConnection(Matrix matrix, int vertice1Row, int
vertice2Row)
        {
            if (matrix.Get(vertice2Row, vertice1Row) == '+')
            {
                return true;
            }

            else
            {
                return false;
            }
        }

        /// <summary>
        /// Checks if the vertice is already in the list and named
        /// </summary>
        /// <param name="vertices">list of all the vertices</param>
        /// <param name="name">name of the needed vertice</param>
        /// <returns>a true or false statement</returns>
        public static bool IsTheVerticeFound(List<Vertice> vertices, string name)
        {
            foreach (Vertice vertice in vertices)
            {
                if (vertice.Name == name)
                {
                    return true;
                }
            }

            return false;
        }

        /// <summary>
        /// Returns a vertice's index by name
        /// </summary>
        /// <param name="vertices">list of all the vertices</param>
```

```
/// <param name="name">name of the vertice</param>
/// <returns>true or false statement</returns>
public static int ReturnIndexByName(List<Vertice> vertices, string name)
{
    for (int i = 0; i < vertices.Count; i++)
    {
        if (vertices[i].Name == name)
        {
            return i;
        }
    }

    return -1;
}

/// <summary>
/// Sorts the list in a custom manner
/// </summary>
/// <param name="vertices">list of all the vertices</param>
public static void CustomListSort(List<Vertice> vertices)
{
    string sting = "Geluonis", tail = "Uodega", waist = "Liemuo";
    Vertice temp = new Vertice();
    for (int i = 0; i < vertices.Count; i++)
    {
        if (vertices[i].Name == sting)
        {
            temp = vertices[0];
            vertices[0] = vertices[i];
            vertices[i] = temp;
        }

        if (vertices[i].Name == tail)
        {
            temp = vertices[1];
            vertices[1] = vertices[i];
            vertices[i] = temp;
        }

        if (vertices[i].Name == waist)
        {
            temp = vertices[2];
            vertices[2] = vertices[i];
            vertices[i] = temp;
        }
    }
}
/// <summary>
/// Checks if the matrix is scorpion
/// </summary>
/// <param name="vertices">list of all the vertices</param>
/// <returns>a true or false statement</returns>
public static bool IsScorpion(List<Vertice> vertices)
{
    foreach (Vertice vertice in vertices)
    {
        if (vertice.Name == null)
        {
            return false;
        }
    }

    return true;
}
```

```
    }
}
```

ErrorProof.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_1_WebApp
{
    public class ErrorProof
    {
        /// <summary>
        /// Checks if there are too many inputs
        /// </summary>
        /// <param name="n">the amount of inputs</param>
        /// <returns>a true or false statement</returns>
        public static bool CheckTheN(int n)
        {
            if (n > 50 || n < 5)
            {
                return false;
            }

            else
            {
                return true;
            }
        }
    }
}
```

Main.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

namespace Lab_1_WebApp
{
    public partial class Main : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            string[] AllLines =
File.ReadAllLines(Server.MapPath(InOutUtils.FormFileName(DropDownList1)));//inputs data
            Matrix scorpionMatrix = InOutUtils.ReadFile(AllLines);
            if (scorpionMatrix == null) //checks if the data is correct
            {
                Label1.Text = "<strong>Neteisingi duomenys!</strong>";
                File.WriteAllText(Server.MapPath("App_Data/Rezultatai.txt"), "Neteisingi
duomenys.");
                return;
```

```
            }

            InOutUtils.StartingDataTable(Table2, scorpionMatrix);
            List<Vertice> AllVertices = new List<Vertice>();

            TaskUtils.FindVertices(scorpionMatrix, 0, AllVertices); //first lists all the
vertices
            TaskUtils.NameVertices(scorpionMatrix, AllVertices); //names them
            TaskUtils.CustomListSort(AllVertices); //sorts them

            bool isScorpion = TaskUtils.IsScorpion(AllVertices); //checks if given data is a
scorpion
            if (!isScorpion)
            {
                InOutUtils.IfError(Label1);
                string[] WrittenLines = InOutUtils.WriteData(scorpionMatrix, AllVertices);
                File.WriteAllLines(Server.MapPath("App_Data/Rezultatai.txt"), WrittenLines);
                File.AppendAllText(Server.MapPath("App_Data/Rezultatai.txt"), "Tai nėra
skorpionas.");
                return;
            }

            else
            {
                string[] WrittenLines = InOutUtils.WriteData(scorpionMatrix, AllVertices);
                InOutUtils.WriteLines(WrittenLines, scorpionMatrix.Rows + 2, AllVertices);
                File.WriteAllLines(Server.MapPath("App_Data/Rezultatai.txt"), WrittenLines);
                InOutUtils.WriteIfScorpion(Label1);
                InOutUtils.FormTable(Table1, AllVertices);
            }
        }

    }
}
```

1.7. Pradiniai duomenys ir rezultatai

```
5
*+-+-
+*---
--*+-
+-+*+
---+*
```

1. U1.txt

SKORPIONO PAIEŠKOS

U1

Išspręsti

Pradiniai duomenys				
5				
*	+	-	+	-
+	*	-	-	-
-	-	*	+	-
+	-	+	*	+
-	-	-	+	*

Matrica yra "skorpionas".

Viršūnės pavadinimas	Viršūnės numeris
Geluonis	2
Uodega	1
Liemuo	4
1 koja	3
2 koja	5

Rezultatai

```
Pradiniai duomenys
n = 5
*+-+-
+*---
--*+-
+-+*+
---+*
Viršūnė: Geluonis    | Numeris: 2
Viršūnė: Uodega      | Numeris: 1
Viršūnė: Liemuo      | Numeris: 4
Viršūnė: 1 koja      | Numeris: 3
Viršūnė: 2 koja      | Numeris: 5
```

Rezultatai.txt

```
6
*--+--
-*+-++
--*+++
+-+*+-
--+-*+
-++-+*
```

2.

U2.txt

SKORPIONO PAIEŠKOS

U2

Išspręsti

Pradiniai duomenys					
6					
*	-	-	+	-	-
-	*	+	-	+	+
-	-	*	+	+	+
+	-	+	*	+	-
-	-	+	-	*	+
-	+	+	-	+	*

Ši matrica nėra "skorpionas".

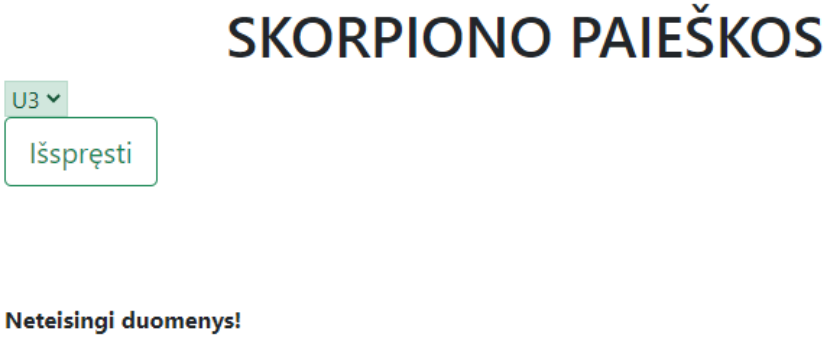
Rezultatai

```
Pradiniai duomenys
n = 6
*--+--
-*+-++
--*+++
+-+*+-
--+-*+
-++-+*
```

```
Tai nėra skorpionas.
```

Rezultatai.txt

3. U3.txt = 51



Rezultatai

```
Neteisingi duomenys.
```

Rezultatai.txt

4.
```
5
*+-+-
+*---
--*+-
+-+*+
---+*
```
U4.txt

SKORPIONO PAIEŠKOS

U4

Išspręsti

Pradiniai duomenys				
5				
*	+	-	+	-
+	*	-	-	-
-	-	*	+	-
+	-	+	*	+
-	-	-	+	*

Matrica yra "skorpionas".

Viršūnės pavadinimas	Viršūnės numeris
Geluonis	2
Uodega	1
Liemuo	4
1 koja	3
2 koja	5

Rezultatai

```
Pradiniai duomenys
n = 5
*+-+-
+*---
--*+-
+-+*+
---+*
Viršūnė: Geluonis    | Numeris: 2
Viršūnė: Uodega      | Numeris: 1
Viršūnė: Liemuo      | Numeris: 4
Viršūnė: 1 koja      | Numeris: 3
Viršūnė: 2 koja      | Numeris: 5
```

Rezultatai.txt

1.8. Dėstytojo pastabos

1. Papildyti programos vartotojo vadovą informacija, kaip sudaromas pradinių duomenų failas, kokiame kataloge jis saugomas, kad vartotojas galėtų naudotis programa su paties sukurtais duomenų rinkiniais.

2. 1.6. skyrelyje turi būti užrašyti klasių pavadinimai, klasės išdėstytos tinkama tvarka.

3. Pradinių duomenų ir rezultatų skyrelyje turi būti pateiktas ne tik vaizdas ekrane, bet ir pradinių duomenų bei rezultatų tekstiniai failai.

(Sutvarkyta)

2. Dinaminis atminties valdymas (L2)

2.1. Darbo užduotis

2.2. Grafinės vartotojo sąsajos schema

2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

2.4. Klasių diagrama

2.5. Programos vartotojo vadovas

2.6. Programos tekstas

2.7. Pradiniai duomenys ir rezultatai

2.8. Dėstytojo pastabos

3. Bendrinės klasės ir testavimas (L3)

3.1. Darbo užduotis

3.2. Grafinės vartotojo sąsajos schema

3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

3.4. Klasių diagrama

3.5. Programos vartotojo vadovas

3.6. Programos tekstas

3.7. Pradiniai duomenys ir rezultatai

3.8. Dėstytojo pastabos

4. Polimorfizmas ir išimčių valdymas (L4)

4.1. Darbo užduotis

4.2. Grafinės vartotojo sąsajos schema

4.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

4.4. Klasių diagrama

4.5. Programos vartotojo vadovas

4.6. Programos tekstas

4.7. Pradiniai duomenys ir rezultatai

4.8. Dėstytojo pastabos

5. Deklaratyvusis programavimas (L5)

5.1. Darbo užduotis

5.2. Grafinės vartotojo sąsajos schema

5.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

5.4. Klasių diagrama

5.5. Programos vartotojo vadovas

5.6. Programos tekstas

5.7. Pradiniai duomenys ir rezultatai

5.8. Dėstytojo pastabos