

Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Augustinas Jukna IFF-0/3

Studentas

Doc. Renata Burbaitė

Dėstytoja

TURINYS

1. Rekursija (L1).....	4
1.1. Darbo užduotis	4
1.2. Grafinės vartotojo sąsajos schema	5
1.3. Sąsajoje panaudotų komponentų keičiamos savybės	5
1.4. Klasių diagrama.....	6
1.5. Programos vartotojo vadovas	6
1.6. Programos tekstas.....	7
1.7. Pradiniai duomenys ir rezultatai	16
1.8. Dėstytojo pastabos.....	21
2. Dinaminis atminties valdymas (L2).....	22
2.1. Darbo užduotis	22
2.2. Grafinės vartotojo sąsajos schema	22
2.3. Sąsajoje panaudotų komponentų keičiamos savybės	23
2.4. Klasių diagrama.....	23
2.5. Programos vartotojo vadovas	24
2.6. Programos tekstas.....	24
2.7. Pradiniai duomenys ir rezultatai	38
2.8. Dėstytojo pastabos.....	43
3. Bendrinės klasės ir testavimas (L3).....	44
3.1. Darbo užduotis	44
3.2. Grafinės vartotojo sąsajos schema	44
3.3. Sąsajoje panaudotų komponentų keičiamos savybės	45
3.4. Klasių diagrama.....	46
3.5. Programos vartotojo vadovas	47
3.6. Programos tekstas.....	47
3.7. Pradiniai duomenys ir rezultatai	68

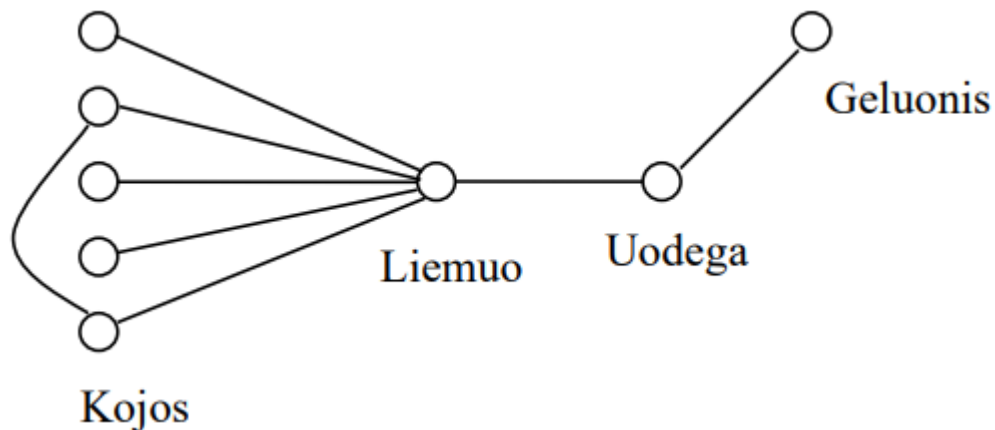
3.8.	Dėstytojo pastabos.....	74
4.	Polimorfizmas ir išimčių valdymas (L4).....	75
4.1.	Darbo užduotis	75
4.2.	Grafinės vartotojo sąsajos schema	75
4.3.	Sąsajoje panaudotų komponentų keičiamos savybės	76
4.4.	Klasių diagrama.....	76
4.5.	Programos vartotojo vadovas	77
4.6.	Programos tekstas.....	77
4.7.	Pradiniai duomenys ir rezultatai.....	100
4.8.	Dėstytojo pastabos.....	101
5.	Deklaratyvusis programavimas (L5).....	108
5.1.	Darbo užduotis	108
5.2.	Grafinės vartotojo sąsajos schema	108
5.3.	Sąsajoje panaudotų komponentų keičiamos savybės	108
5.4.	Klasių diagrama.....	108
5.5.	Programos vartotojo vadovas	108
5.6.	Programos tekstas.....	108
5.7.	Pradiniai duomenys ir rezultatai.....	108
5.8.	Dėstytojo pastabos.....	109

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 (liemenu). 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“.



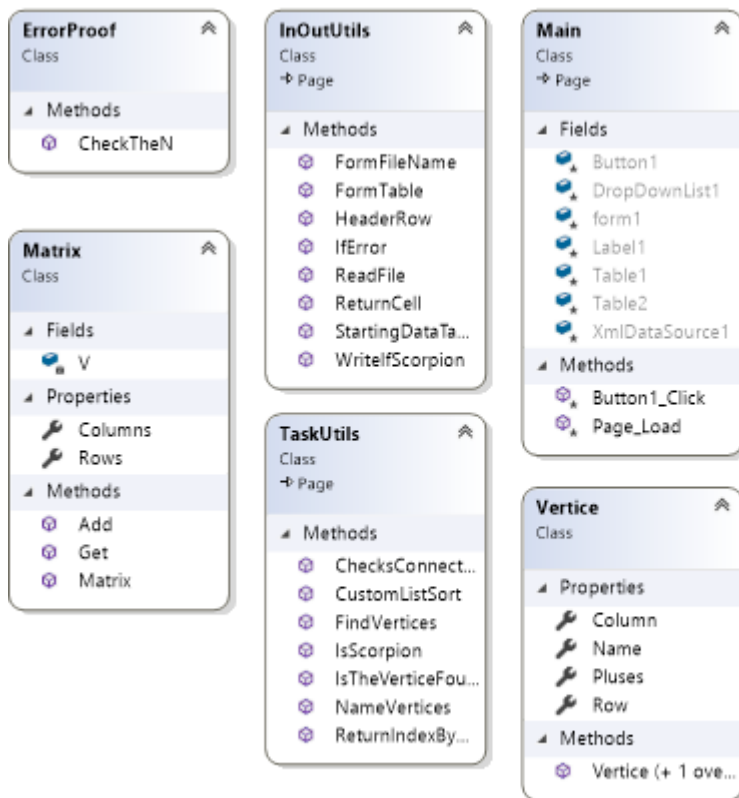
1.2. Grafinės vartotojo sąsajos schema



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:
- Turi ne daugiau nei 50 ar ne mažiau nei 5 viršūnės;
 - turi vieną geluonies viršūnę, kuri jungiasi su uodega;
 - uodegą, kuri jungiasi su geluonimi ir liemeniu;
 - liemenį, kuris jungiasi su uodega bei viena ar daugiau kojų;
 - 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 = "Liemu";
    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.WriteLine(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

SKORPIONO PAIEŠKOS

U3 ▾

Išspręsti

Neteisingi duomenys!

Rezultatai

Neteisingi duomenys.

Rezultatai.txt

4.

U4.txt

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

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 pradinų 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. Pradinų duomenų ir rezultatų skyrelyje turi būti pateiktas ne tik vaizdas ekrane, bet ir pradinų duomenų bei rezultatų tekstiniai failai.

(Sutvarkyta)

2. Dinaminis atminties valdymas (L2)

2.1. Darbo užduotis

LD_8. Maršrutai.

Turizmo agentūra rengia kelionę po Lietuvą iš nurodyto miesto. Išvykus iš vieno miesto galima nukeliauti į bet kurį kitą miestą. Tarp miestų gali būti daugiau kaip vienas kelionės maršrutas. Kelionės metu tas pats miestas gali būti aplankytas tik vieną kartą ir galima lankyti tik tuos miestus, kuriuose gyventojų skaičius yra mažesnis už nurodytą. Maršrutas nebūtinai turi apimti visus nurodytus miestus. Reikia parašyti programą, kuri pasiūlytų kelionės maršrutus, kurių ilgis viršija nurodytą (įvedama klaviatūra).

Duomenys:

- Leidžiamas lankyti gyventojų skaičius ir miestas, iš kur prasideda kelionė, nurodomi klaviatūra. • Tekstiniame faile U8a.txt yra duomenys apie kelius tarp miestų. Failo eilutėse surašyta: pirmojo miesto pavadinimas, antrojo miesto pavadinimas, kelio tarp pirmojo ir antrojo miesto ilgis kilometrais. Miesto pavadinimas gali būti iš dviejų žodžių.
- Tekstiniame faile U8b.txt yra duomenys apie miestų gyventojų skaičius. Bus visi miestai, paminėti U8a.txt. Failo eilutėje yra informacija apie vieną miestą: miesto pavadinimas, miesto gyventojų skaičius. Spausdinamas sąrašas turi būti surikiuotas pagal maršruto ilgį ir pirmojo miesto pavadinimą. Realizuokite netinkamų maršrutų (miestas, kurio nenorite aplankyti, įvedamas klaviatūra) pašalinimą iš sąrašo.

2.2. Grafinės vartotojo sąsajos schema

MARŠRUTŲ PAIEŠKOS

Įveskite pradinį miestą:

Reikalingų pradinio duomenų įvedimas

Įveskite didžiausią leistiną gyventojų kiekį mieste:

Įveskite minimalų maršruto atstumą kilometrais:

Surasti

←

Maršrutų suradimo mygtukas

Miestas, kurio nenorite aplankyti:

Ištrinti

←

Ištrina nepageidaujamo miesto maršrutus

Pradinio duomenų bei rezultatų lentelės

Pradiniai duomenys (maršrutai)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Kaunas	Kėdainiai	56
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Kaunas	Panevėžys	108
Kaunas	Vilnius	110

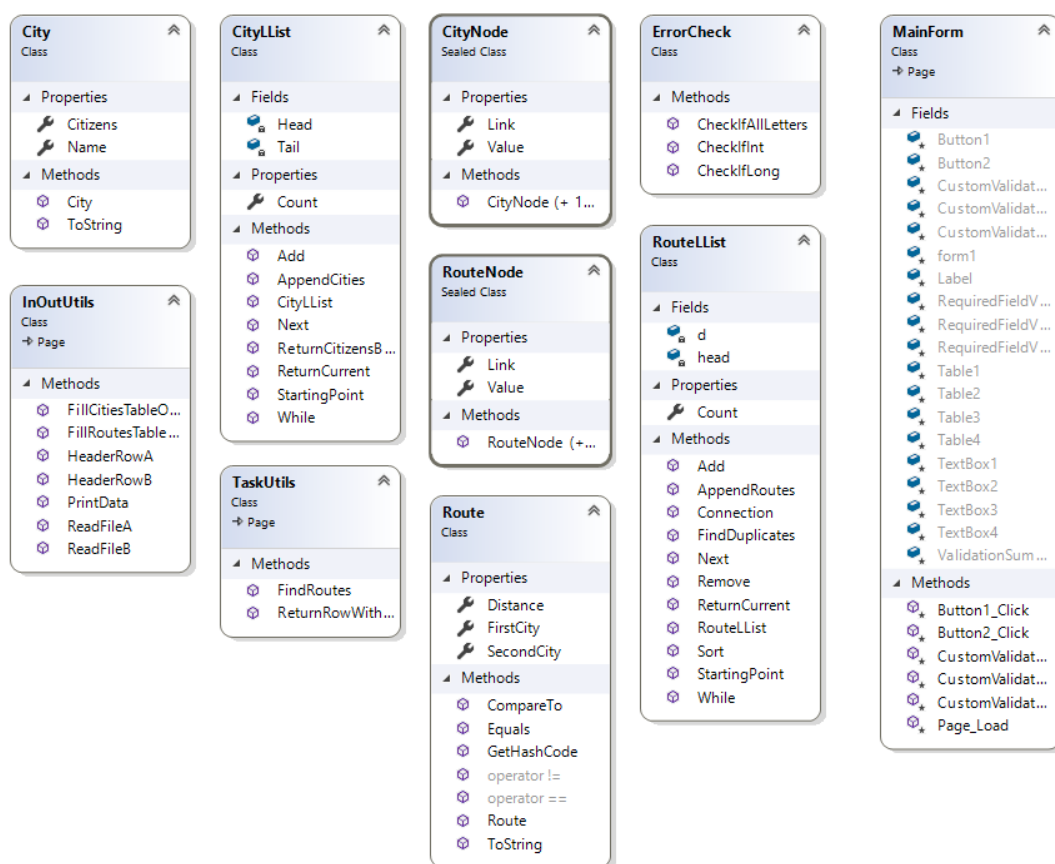
Pradiniai duomenys (miestai)	
Miesto pavadinimas	Gyventojų kiekis
Kaunas	295269
Vilnius	544386
Panevėžys	85885
Klaipėda	152818
Kėdainiai	22677
Švenčionys	4065
Marijampolė	34975
Šiauliai	101511
Plungė	23246

Rezultatai

2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
#TextBox1, #TextBox2, #TextBox3, #TextBox4	Height	35px
#TextBox1, #TextBox2, #TextBox3, #TextBox4	Width	300px
#TextBox4, #Button2, #Label, #Table1, #Table2, #Table3, #Table4	Visible	false
#TextBox1, #TextBox2, #TextBox3, #TextBox4	CssClass	margin-bottom: 17px;
#Label	CssClass	margin-top: 40px;
#header	CssClass	Margin-top:30px;
#ValidationSummary1	CssClass	alert alert-danger
#ValidationSummary1	ForeColor	Red
#Button1, #Button2	CssClass	btn btn-light

2.4. Klasių diagrama



2.5. Programos vartotojo vadovas

- ✓ Atsidarius puslapiui, vartotoją pasitinką kelios įvestys bei mygtukas „Surasti“.
- ✓ Norint, jog programa pradėtų veikti, reikia įvesti:
 - Miestą, nuo kurio prasidės visas maršrutas;
 - Didžiausią leistiną gyventojų skaičių mieste, kuris bus maršruto dalimi;
 - Minimalų maršruto atstumą.
- ✓ Paspaudus mygtuką „Surasti“, atsiras trys lentelės. Pirmoje bus pavaizduota pradiniai maršrutų duomenys, antroje – miestų, o trečioje – jau atrinkti maršrutai vartotojui.
- ✓ Taip pat, atsiranda dar vienas naujas įvesties laukelis – į jį galima įvesti miestą, kurio programos naudotojas nenorėtų aplankyti. Tas miestas bus pašalintas iš galutinio sąrašo bei lentelės.
- ✓ Norint suformuoti pradinius duomenis, reikia 2 pradinių duomenų dokumentų – „U8a.txt“ bei „U8b.txt“. „U8a.txt“ duomenys išdėstomi tokia tvarka: pirma rašomas pirmojo miesto pavadinimas, antru numeriu – antrojo miesto, o trečia įvestis eilutėje – atstumas tarp šių dviejų miestų. Duomenys yra atskiriami kabliataškiais („;“). Tokia seka, pasirinkus naują eilutę yra taip pat tęsiama. „U8b.txt“ duomenyse yra aprašomi anksčiau minėto dokumento miestų gyventojų skaičiai. Duomenys išdėstyti tokia tvarka – pirma eina miesto pavadinimas, po to – gyventojų kiekis tame mieste. Viskas taip pat yra skiriama kabliataškiu. Abu dokumentai yra randami serverio „App_Data“ aplanke.

2.6. Programos tekstas

City.cs

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

namespace LD2_WebApp
{
    class City
    {
        public string Name { get; set; }
        public long Citizens { get; set; }

        public City(string city, long citizens)
        {
            this.Name = city;
            this.Citizens = citizens;
        }
        /// <summary>
        /// ToString method override
    }
}
```



```

    /// </summary>
    public override string ToString()
    {
        string line = String.Format("|{0, -20}|{1, 20}|", Name, Citizens);
        return line;
    }
}
}

```

CityNode.cs

```

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

namespace LD2_WebApp
{
    sealed class CityNode
    {
        public City Value { get; set; }
        public CityNode Link { get; set; }
        public CityNode() { } //empty constructor
        public CityNode(City value, CityNode link) //constructor with two variables
        {
            this.Value = value;
            this.Link = link;
        }
    }
}

```

CityLList.cs

```

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

namespace LD2_WebApp
{
    class CityLList
    {
        private CityNode Head;
        private CityNode Tail;
        public int Count { get; private set; }

        public CityLList()
        {
            this.Head = null;
            this.Tail = null;
            this.Count = 0;
        }

        /// <summary>
        /// Adds a new element (City class object) to the linked list
        /// </summary>
        /// <param name="city">object to add</param>
        public void Add(City city)
        {
            CityNode newNode = new CityNode(city, null);
            if (this.Head == null)
            {

```

```

        this.Head = newNode;
        this.Tail = newNode;
    }

    else
    {
        this.Tail.Link = newNode;
        this.Tail = newNode;
    }
    Count++;
}

/// <summary>
/// Sets tail to the starting point - head
/// </summary>
public void StartingPoint()
{
    this.Tail = this.Head;
}

/// <summary>
/// Forces nodes to move
/// </summary>
public void Next()
{
    this.Tail = this.Tail.Link;
}

/// <summary>
/// Keeps in check if the tail is equal to null or not
/// </summary>
/// <returns></returns>
public bool While()
{
    return this.Tail != null;
}

/// <summary>
/// Returns the current tail's value
/// </summary>
/// <returns></returns>
public City ReturnCurrent()
{
    return this.Tail.Value;
}

/// <summary>
/// Returns the amount of citizens by city name
/// </summary>
/// <param name="cityName"></param>
/// <returns>amount of citizens in the city</returns>
public long ReturnCitizensByName(string cityName)
{
    for (CityNode w = this.Tail; w != null; w = w.Link)
    {
        if (w.Value.Name == cityName)
        {
            return w.Value.Citizens;
        }
    }
    return 0;
}

/// <summary>
/// Appends all cities from the linked list to a string array
/// </summary>
/// <param name="AllLines">name of the array</param>

```

```

        /// <param name="index">index which shows in which array's place to put data
in</param>
        public void AppendCities(string[] AllLines, ref int index)
        {
            if (Count == 0)
            {
                AllLines[index++] = "Miestų nėra.";
                return;
            }

            AllLines[index++] = String.Format("{0, -20}|{1, 20}", "Miesto pavadinimas",
"Gyventojų kiekis");
            for (CityNode w = Head; w != null; w = w.Link)
            {
                AllLines[index++] = w.Value.ToString();
            }
            AllLines[index++] = String.Format("");
        }
    }
}

```

Route.cs

```

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

namespace LD2_WebApp
{
    class Route
    {
        public string FirstCity { get; set; }
        public string SecondCity { get; set; }
        public int Distance { get; set; }

        public Route(string firstCity, string secondCity, int distance)//constructor
        {
            this.FirstCity = firstCity;
            this.SecondCity = secondCity;
            this.Distance = distance;
        }

        /// <summary>
        /// CompareTo method override
        /// </summary>
        /// <param name="route">another Route object to compare</param>
        /// <returns>CompareTo result (-1, 0 or 1)</returns>
        public int CompareTo(Route route)
        {
            if (this.Distance == route.Distance)
            {
                return this.FirstCity.CompareTo(route.FirstCity);
            }

            else
            {
                return this.Distance.CompareTo(route.Distance);
            }
        }

        /// <summary>
        /// Override for == operator
        /// </summary>

```

```

    /// <param name="a">One class object</param>
    /// <param name="b">Another class object</param>
    /// <returns>a true or false statement</returns>
    public static bool operator == (Route a, Route b)
    {
        return a.FirstCity == b.FirstCity && a.SecondCity == b.SecondCity && a.Distance ==
b.Distance;
    }

    /// <summary>
    /// Override for != operator
    /// </summary>
    /// <param name="a">One class object</param>
    /// <param name="b">Another class object</param>
    /// <returns>a true or false statement</returns>
    public static bool operator != (Route a, Route b)
    {
        return a.FirstCity != b.FirstCity && a.SecondCity != b.SecondCity && a.Distance !=
b.Distance;
    }

    /// <summary>
    /// Override for Equals() method
    /// </summary>
    /// <param name="obj">Another class object</param>
    /// <returns>a true or false statement</returns>
    public override bool Equals(object obj)
    {
        Route route = obj as Route;
        return FirstCity.Equals(route.FirstCity) && SecondCity.Equals(route.SecondCity) &&
Distance.Equals(route.Distance);
    }

    /// <summary>
    /// GetHashCode() method override
    /// </summary>
    /// <returns></returns>
    public override int GetHashCode()
    {
        return FirstCity.GetHashCode() ^ SecondCity.GetHashCode() ^
Distance.GetHashCode();
    }

    /// <summary>
    /// ToString() method override
    /// </summary>
    /// <returns>a line of string</returns>
    public override string ToString()
    {
        string line = String.Format("|{0, -20}|{1, -20}|{2, 15}", FirstCity, SecondCity,
Distance);
        return line;
    }
}

```

RouteNode.cs

```

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

namespace LD2_WebApp
{

```

```

sealed class RouteNode
{
    public Route Value { get; set; }
    public RouteNode Link { get; set; }
    public RouteNode() { } //empty constructor
    public RouteNode(Route value, RouteNode link) //cosntructor with two variables
    {
        this.Value = value;
        this.Link = link;
    }
}
}

```

RouteLList.cs

```

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

namespace LD2_WebApp
{
    class RouteLList
    {
        private RouteNode head;
        private RouteNode d;
        public int Count { get; private set; } //count to have an easy and accessible way to
        check the count of list items

        public RouteLList() //constructor
        {
            this.head = null;
            this.Count = 0;
        }

        /// <summary>
        /// Adds a class object to the list (creates a new node)
        /// </summary>
        /// <param name="route">Route class object to add</param>
        public void Add(Route route)
        {
            head = new RouteNode(route, head);
            Count++;
        }

        /// <summary>
        /// Sets pointer d to a starting point
        /// </summary>
        public void StartingPoint() => d = head;

        /// <summary>
        /// Checks if a pointer is equal to null or not
        /// </summary>
        /// <returns>true or false</returns>
        public bool While()
        {
            return d != null;
        }

        /// <summary>
        /// Makes pointer move forward in the list
        /// </summary>
        public void Next() => d = d.Link;
    }
}

```

```

/// <summary>
/// Returns the value in which pointer is pointing
/// </summary>
/// <returns>value of pointer node</returns>
public Route ReturnCurrent()
{
    return this.d.Value;
}

/// <summary>
/// Sorts this linked list
/// </summary>
public void Sort()
{
    RouteNode a = this.head;
    while (a != null)
    {
        RouteNode b = a;
        RouteNode min = a;
        while (b != null)
        {
            if (min.Value.CompareTo(b.Value) > 0)
            {
                min = b;
            }
            b = b.Link;
        }
        Route temp = a.Value;
        a.Value = min.Value;
        min.Value = temp;
        a = a.Link;
    }
}

/// <summary>
/// Removes a specific object from the list
/// </summary>
/// <param name="cityName">name of the objects first or second city</param>
public void Remove(string cityName)
{
    RouteNode current = head;
    while (current != null)
    {
        if ((current.Value.FirstCity == cityName || current.Value.SecondCity ==
cityName) && current == head)
        {
            head = head.Link;
        }

        else if ((current.Value.FirstCity == cityName || current.Value.SecondCity ==
cityName))
        {
            RouteNode j;
            for (j = head; j.Link != current; j = j.Link); //finds the previous node
            j.Link = current.Link;
        }

        current = current.Link;
    }
}

/// <summary>
/// Checks if there are no duplicates in the list
/// </summary>
/// <param name="w">Route object to check the list for</param>
/// <returns>true or false</returns>

```

```

public bool FindDuplicates(Route w)
{
    for (RouteNode temp = this.d; temp != null; temp = temp.Link)
    {
        if (temp.Value == w)
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Checks a connection between cities (if for example: starting city is Kaunas,
/// </summary>                                     //route.FirstCity == "Kaunas";
route.SecondCity == "Vilnius"; There is a connection between them
/// <param name="startingCity">user's inputed starting city</param>
/// <param name="route">Route class object </param>
/// <returns>true or false</returns>
public bool Connection(string startingCity, Route route)
{
    for (RouteNode w = head; w != null; w = w.Link)
    {
        if (w.Value.FirstCity == startingCity && w.Value.SecondCity ==
route.FirstCity)
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Appends all the routes from the list
/// </summary>
/// <param name="AllLines">string array which holds all the lines to print</param>
/// <param name="index">array's place to assign variables to</param>
public void AppendRoutes(string[] AllLines, ref int index)
{
    if (Count == 0)
    {
        AllLines[index++] = "Tokių maršrutų nėra.";
        return;
    }

    AllLines[index++] = String.Format("|{0, -20}|{1, -20}|{2, 15}", "Pirmas miestas",
"Antras miestas", "Atstumas");
    for (RouteNode w = head; w != null; w = w.Link)
    {
        AllLines[index++] = w.Value.ToString();
    }
    AllLines[index++] = String.Format("");
}
}
}

```

InOutUtils.cs

```

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

```

```

using System.IO;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace LD2_WebApp
{
    class InOutUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Reads the input file's data (U8a.txt)
        /// </summary>
        /// <param name="AllLines">string array which holds file's data</param>
        /// <returns>a made list of RouteLList </returns>
        public static RouteLList ReadFileA(string[] AllLines)
        {
            RouteLList routeList = new RouteLList();
            foreach (string line in AllLines)
            {
                string[] AllParts = line.Split(';');
                string cityA = AllParts[0];
                string cityB = AllParts[1];
                int distance = int.Parse(AllParts[2]);
                Route route = new Route(cityA, cityB, distance);
                routeList.Add(route);
            }
            return routeList;
        }

        /// <summary>
        /// Reads the input file's data (U8b.txt)
        /// </summary>
        /// <param name="AllLines">string array which holds all the data from the file</param>
        /// <returns>returns CityLList object</returns>
        public static CityLList ReadFileB(string[] AllLines)
        {
            CityLList cityList = new CityLList();
            foreach (string line in AllLines)
            {
                string[] AllParts = line.Split(';');
                string name = AllParts[0];
                long citizens = long.Parse(AllParts[1]);
                City city = new City(name, citizens);
                cityList.Add(city);
            }
            return cityList;
        }

        /// <summary>
        /// Fills RouteLList table on screen
        /// </summary>
        /// <param name="table">table to modify</param>
        /// <param name="filler">list to take data from</param>
        public static void FillRoutesTableOnScreen(Table table, RouteLList filler)
        {
            table.Rows.Add(HeaderRowA());
            for (filler.StartingPoint(); filler.While(); filler.Next())
            {
                TableRow row = new TableRow();
                Route temp = filler.ReturnCurrent();
                row.Cells.Add(new TableCell { Text = temp.FirstCity });
                row.Cells.Add(new TableCell { Text = temp.SecondCity });
                row.Cells.Add(new TableCell { Text = (temp.Distance).ToString() });
                table.Rows.Add(row);
            }
        }
    }
}

```



```

/// <summary>
/// Fills CityLList table on screen
/// </summary>
/// <param name="table">table to modify</param>
/// <param name="filler">CityLList object to take data from</param>
public static void FillCitiesTableOnScreen(Table table, CityLList filler)
{
    table.Rows.Add(HeaderRowB());
    for (filler.StartingPoint(); filler.While(); filler.Next())
    {
        TableRow row = new TableRow();
        City temp = filler.ReturnCurrent();
        row.Cells.Add(new TableCell { Text = temp.Name });
        row.Cells.Add(new TableCell { Text = (temp.Citizens).ToString() });
        table.Rows.Add(row);
    }
}

/// <summary>
/// Header row for Route class objects
/// </summary>
/// <returns>a header row for the table</returns>
public static TableRow HeaderRowA()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell { Text = "Pirmas miestas" });
    row.Cells.Add(new TableCell { Text = "Antras miestas" });
    row.Cells.Add(new TableCell { Text = "Atstumas tarp miestų" });
    return row;
}

/// <summary>
/// Header row for City class objects
/// </summary>
/// <returns>a header row for the table</returns>
public static TableRow HeaderRowB()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell { Text = "Miesto pavadinimas" });
    row.Cells.Add(new TableCell { Text = "Gyventojų kiekis" });
    return row;
}

/// <summary>
/// Combines all the data and "prints" it into a string array
/// </summary>
/// <param name="start1">list of all the starting data (Route objects)</param>
/// <param name="start2">list of all the starting data (City objects)</param>
/// <param name="end">list of filtered Route class objects</param>
/// <returns>a string array</returns>
public static string[] PrintData(RouteLList start1, CityLList start2, RouteLList end)
{
    string[] AllLines = new string[start1.Count + start2.Count + end.Count + 10];
    int index = 0;
    AllLines[index++] = String.Format("Pradiniai duomenys");
    start1.AppendRoutes(AllLines, ref index);
    start2.AppendCities(AllLines, ref index);
    AllLines[index++] = String.Format("Rezultatai");
    end.AppendRoutes(AllLines, ref index);

    return AllLines;
}
}
}

```

TaskUtils.cs

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

namespace LD2_WebApp
{
    class TaskUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Main method of the whole program. Finds all the route combinations which fit the
        requirements
        /// </summary>
        /// <param name="startingCity">user's inputed starting city</param>
        /// <param name="maxCitizens">max citizens in a city</param>
        /// <param name="minDistance">minimum distance in a route</param>
        /// <param name="AllRoutes">list of all the routes</param>
        /// <param name="AllCities">list of all the cities</param>
        /// <returns></returns>
        public static RouteLList FindRoutes(string startingCity, long maxCitizens, int
minDistance, RouteLList AllRoutes, CityLList AllCities)
        {
            RouteLList possibleRoutes = new RouteLList();
            RouteLList w = AllRoutes;
            for (w.StartingPoint(); w.While(); w.Next())
            {
                if (((w.ReturnCurrent().FirstCity == startingCity &&
w.ReturnCurrent().Distance >= minDistance) || (w.ReturnCurrent().FirstCity != startingCity &&
AllRoutes.Connection(startingCity, w.ReturnCurrent()) && w.ReturnCurrent().Distance >=
minDistance && AllCities.ReturnCitizensByName(w.ReturnCurrent().FirstCity) <= maxCitizens
&& AllCities.ReturnCitizensByName(w.ReturnCurrent().SecondCity) <=
maxCitizens)) && !possibleRoutes.FindDuplicates(w.ReturnCurrent()))
                {
                    possibleRoutes.Add(w.ReturnCurrent());
                }
            }
            return possibleRoutes;
        }

        /// <summary>
        /// Returns a row with specific test
        /// </summary>
        /// <param name="text">text to put in a row</param>
        /// <param name="n">amount of cell's column span</param>
        /// <returns></returns>
        public static TableRow ReturnRowWithText(string text, int n)
        {
            TableRow row = new TableRow();
            row.Cells.Add(new TableCell { Text = text, ColumnSpan = n });
            return row;
        }
    }
}
```

ErrorCheck.cs

```
using System;
using System.Collections.Generic;
```

```

using System.Linq;
using System.Web;

namespace LD2_WebApp
{
    public class ErrorCheck
    {
        /// <summary>
        /// Checks if a given string parses to long or not
        /// </summary>
        /// <param name="value">string to parse from</param>
        /// <returns>a true or false statement</returns>
        public static bool CheckIfLong(string value)
        {
            bool c = long.TryParse(value, out long number);
            return c;
        }

        /// <summary>
        /// Checks if a given string parses into integer
        /// </summary>
        /// <param name="value">name of the string to parse from</param>
        /// <returns>a true or false statement</returns>
        public static bool CheckIfInt(string value)
        {
            bool c = int.TryParse(value, out int number);
            return c;
        }

        /// <summary>
        /// Checks if the string contains only letters and no special symbols
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public static bool CheckIfAllLetters(string value)
        {
            int count = 0;
            for (int i = 0; i < value.Count(); i++)
            {
                if (char.IsLetter(value[i]))
                {
                    count++;
                }
            }
            if (count == value.Count())
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}

```

MainForm.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 LD2_WebApp
{
    public partial class MainForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Page.IsPostBack && Session["TABLE1"] != null && Session["TABLE2"] != null &&
Session["TABLE3"] != null && Table1.Rows.Count == 0 && Table2.Rows.Count == 0 &&
Table3.Rows.Count == 0)
            {
                Table1.Rows.AddRange(((TableRow[])Session["TABLE1"]));
                Table2.Rows.AddRange(((TableRow[])Session["TABLE2"]));
                Table3.Rows.AddRange(((TableRow[])Session["TABLE3"]));
            }
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                const string CFdA = "App_Data/U8a.txt";
                const string CFdB = "App_Data/U8b.txt";
                const string CFr = "App_Data/Rezultatai.txt";

                if (!File.Exists(Server.MapPath(CFdA)) || !File.Exists(Server.MapPath(CFdB)))
                {
                    Session["Files"] = false;
                    Page.Validate();
                    return;
                }

                string[] AllLinesA = File.ReadAllLines(Server.MapPath(CFdA));
                string[] AllLinesB = File.ReadAllLines(Server.MapPath(CFdB));

                RouteLList AllRoutes = InOutUtils.ReadFileA(AllLinesA);
                CityLList AllCities = InOutUtils.ReadFileB(AllLinesB);

                string startingCity = TextBox1.Text;
                long maxCitizens = long.Parse(TextBox2.Text);
                int minDistance = int.Parse(TextBox3.Text);

                RouteLList FilteredRoutes = TaskUtils.FindRoutes(startingCity, maxCitizens,
minDistance, AllRoutes, AllCities);
                FilteredRoutes.Sort();
                Table1.Rows.Clear();
                Table2.Rows.Clear();
                Table3.Rows.Clear();

                Table1.Rows.Add(TaskUtils.ReturnRowWithText("Pradiniai duomenys (maršrutai)",
3));
                InOutUtils.FillRoutesTableOnScreen(Table1, AllRoutes);

                Table2.Rows.Add(TaskUtils.ReturnRowWithText("Pradiniai duomenys (miestai)",
2));
                InOutUtils.FillCitiesTableOnScreen(Table2, AllCities);

                Table3.Rows.Add(TaskUtils.ReturnRowWithText("Rezultatai", 3));
                InOutUtils.FillRoutesTableOnScreen(Table3, FilteredRoutes);

                Table1.Visible = true;
                Table2.Visible = true;
                Table3.Visible = true;

                string[] AllLines = InOutUtils.PrintData(AllRoutes, AllCities,
FilteredRoutes);

                if (File.Exists(Server.MapPath(CFr)))

```

```

    {
        File.Delete(Server.MapPath(CFr));
    }

    File.AppendAllLines(Server.MapPath(CFr), AllLines);

    Button2.Visible = true;
    TextBox4.Visible = true;
    Label1.Visible = true;

    TableRow[] rows1 = new TableRow[Table1.Rows.Count];
    Table1.Rows.CopyTo(rows1, 0);
    Session.Remove("TABLE1");
    Session.Add("TABLE1", rows1);

    TableRow[] rows2 = new TableRow[Table2.Rows.Count];
    Table2.Rows.CopyTo(rows2, 0);
    Session.Remove("TABLE2");
    Session.Add("TABLE2", rows2);

    TableRow[] rows3 = new TableRow[Table3.Rows.Count];
    Table3.Rows.CopyTo(rows3, 0);
    Session.Remove("TABLE3");
    Session.Add("TABLE3", rows3);

    Session["RouteList"] = FilteredRoutes;
    Session["CFrAddress"] = CFr;
}
}

protected void Button2_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string CFr = (string)Session["CFrAddress"];
        string removeCity = TextBox4.Text;
        RouteLList filtered = (RouteLList)Session["RouteList"];
        filtered.Remove(removeCity);
        Table4.Rows.Add(TaskUtils.ReturnRowWithText("Rezultatai (po panaikinimo)",
3));
        InOutUtils.FillRoutesTableOnScreen(Table4, filtered);
        Table4.Visible = true;

        string[] AllLines = new string[filtered.Count + 3];
        int index = 0;
        AllLines[index++] = "Rezultatai (po panaikinimo)";
        filtered.AppendRoutes(AllLines, ref index);
        File.AppendAllLines(Server.MapPath(CFr), AllLines);
    }
}

protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!ErrorCheck.CheckIfLong(TextBox2.Text))
    {
        args.IsValid = false;
    }
    else
    {
        args.IsValid = true;
    }
}

```

```

        protected void CustomValidator2_ServerValidate(object source, ServerValidateEventArgs
args)
        {
            if (!ErrorCheck.CheckIfInt(TextBox3.Text))
            {
                args.IsValid = false;
            }

            else
            {
                args.IsValid = true;
            }
        }

        protected void CustomValidator3_ServerValidate(object source, ServerValidateEventArgs
args)
        {
            if (!ErrorCheck.CheckIfAllLetters(TextBox1.Text))
            {
                args.IsValid = false;
            }

            else
            {
                args.IsValid = true;
            }
        }

        protected void CustomValidator4_ServerValidate(object source, ServerValidateEventArgs
args)
        {
            if (!ErrorCheck.CheckIfAllLetters(TextBox4.Text))
            {
                args.IsValid = false;
            }

            else
            {
                args.IsValid = true;
            }
        }

        protected void CustomValidator5_ServerValidate(object source, ServerValidateEventArgs
args)
        {
            if (Session["Files"] != null && !(bool)Session["Files"])
            {
                args.IsValid = false;
            }

            else
            {
                args.IsValid = true;
            }
        }
    }
}

```

2.7. Pradiniai duomenys ir rezultatai

1) variantas

U8a.txt

Kaunas;Vilnius;110

Kaunas;Panevėžys;108

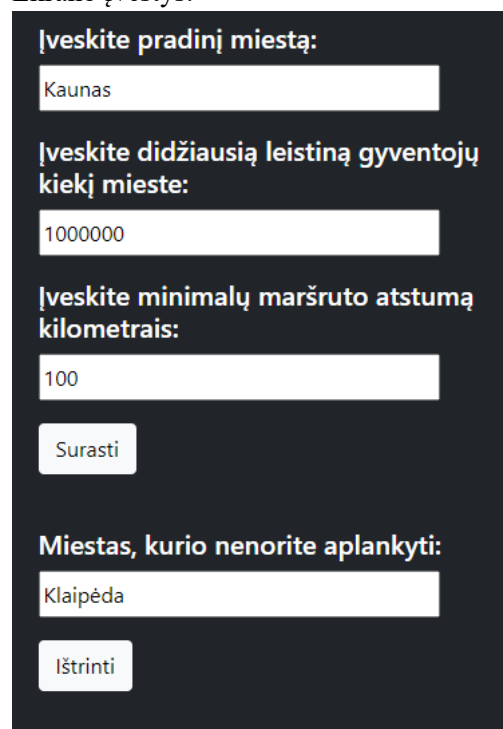
Kaunas;Klaipėda;214

Kaunas;Švenčionys;182
Kaunas;Kėdainiai;56
Vilnius;Klaipėda;286
Marijampolė;Klaipėda;286
Šiauliai;Plungė;102

U8b.txt

Kaunas;295269
Vilnius;544386
Panevėžys;85885
Klaipėda;152818
Kėdainiai;22677
Švenčionys;4065
Marijampolė;34975
Šiauliai;101511
Plungė;23246

Ekrano įvestys:



The screenshot shows a web form with a dark background and white text. It contains three input fields with labels in Lithuanian, a 'Surasti' (Find) button, and another input field with a 'Ištrinti' (Delete) button.

Įveskite pradinį miestą:
Kaunas

Įveskite didžiausią leistiną gyventojų
kiekį mieste:
1000000

Įveskite minimalų maršruto atstumą
kilometrais:
100

Surasti

Miestas, kurio nenorite aplankyti:
Klaipėda

Ištrinti

Rezultatai.txt

Rezultatai - Notepad		
File Edit Format View Help		
Pradiniai duomenys		
Pirmas miestas	Antras miestas	Atstumas
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Kaunas	Kėdainiai	56
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Miesto pavadinimas	Gyventojų kiekis	
Kaunas	295269	
Vilnius	544386	
Panevėžys	85885	
Klaipėda	152818	
Kėdainiai	22677	
Švenčionys	4065	
Marijampolė	34975	
Šiauliai	101511	
Plungė	23246	
Rezultatai		
Pirmas miestas	Antras miestas	Atstumas
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Vilnius	Klaipėda	286
Rezultatai (po panaikinimo)		
Pirmas miestas	Antras miestas	Atstumas
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182

Ekranų rezultatai

Pradiniai duomenys (maršrutai)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Kaunas	Kėdainiai	56
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Kaunas	Panevėžys	108
Kaunas	Vilnius	110

Pradiniai duomenys (miestai)	
Miesto pavadinimas	Gyventojų kiekis
Kaunas	295269
Vilnius	544386
Panevėžys	85885
Klaipėda	152818
Kėdainiai	22677
Švenčionys	4065
Marijampolė	34975
Šiauliai	101511
Plungė	23246

Rezultatai		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Vilnius	Klaipėda	286

Rezultatai (po panaikinimo)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182

2 variantas)

U8a.txt

Vilnius;Klaipėda;286

Marijampolė;Klaipėda;286

Šiauliai;Plungė;102

U8b.txt

Vilnius;100000

Klaipėda;1552233

Marijampolė;15522

Šiauliai;1545452

Plungė;155222

Rezultatai.txt

Rezultatai - Notepad		
File Edit Format View Help		
Pradiniai duomenys		
Pirmas miestas	Antras miestas	Atstumas
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Miesto pavadinimas	Gyventojų kiekis	
Vilnius	100000	
Klaipėda	1552233	
Marijampolė	15522	
Šiauliai	1545452	
Plungė	155222	
Rezultatai		
Tokių maršrutų nėra.		
Rezultatai (po panaikinimo)		
Tokių maršrutų nėra.		

Ekrano įvestys/rezultatai

MARŠRUTŲ PAIEŠKOS

Įveskite pradinį miestą:

Kaunas

Įveskite didžiausią leistiną gyventojų kiekį mieste:

100000

Įveskite minimalų maršruto atstumą kilometrais:

100

Surasti

Miestas, kurio nenorite aplankyti:

Kaunas

Ištrinti

Pradiniai duomenys (maršrutai)

Pirmas miestas	Antras miestas	Atstumas tarp miestų
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286

Pradiniai duomenys (miestai)

Miesto pavadinimas	Gyventojų kiekis
Vilnius	100000
Klaipėda	1552233
Marijampolė	15522
Šiauliai	1545452
Plungė	155222

Rezultatai

Maršrutų nėra.

Rezultatai (po panaikinimo)

Maršrutų nėra.

3 variantas)

Nėra vieno ar abiejų duomenų failų.

MARŠRUTŲ PAIEŠKOS

Trūksta duomenų failo/failų!

[veskite pradinį miestą:

Kaunas

[veskite didžiausią leistiną gyventojų kiekį mieste:

100000

[veskite minimalų maršruto atstumą kilometrais:

100

Surasti

2.8. Dėstytojo pastabos

Testas: 2/3

Gynimas: 6/6

Ataskaita: 1/1

Galutinis įvertinimas: 9.

3. Bendrinės klasės ir testavimas (L3)

3.1. Darbo užduotis

LD_8. Maršrutai.

Turizmo agentūra rengia kelionę po Lietuvą iš nurodyto miesto. Išvykus iš vieno miesto galima nukeliauti į bet kurį kitą miestą. Tarp miestų gali būti daugiau kaip vienas kelionės maršrutas. Kelionės metu tas pats miestas gali būti aplankytas tik vieną kartą ir galima lankyti tik tuos miestus, kuriuose gyventojų skaičius yra mažesnis už nurodytą. Maršrutas nebūtinai turi apimti visus nurodytus miestus. Reikia parašyti programą, kuri pasiūlytų kelionės maršrutus, kurių ilgis viršija nurodytą (įvedama klaviatūra).

Duomenys:

- Leidžiamas lankyti gyventojų skaičius ir miestas, iš kur prasideda kelionė, nurodomi klaviatūra. • Tekstiniame faile U8a.txt yra duomenys apie kelius tarp miestų. Failo eilutėse surašyta: pirmojo miesto pavadinimas, antrojo miesto pavadinimas, kelio tarp pirmojo ir antrojo miesto ilgis kilometrais. Miesto pavadinimas gali būti iš dviejų žodžių.
- Tekstiniame faile U8b.txt yra duomenys apie miestų gyventojų skaičius. Bus visi miestai, paminėti U8a.txt. Failo eilutėje yra informacija apie vieną miestą: miesto pavadinimas, miesto gyventojų skaičius. Spausdinamas sąrašas turi būti surikiuotas pagal maršruto ilgį ir pirmojo miesto pavadinimą. Realizuokite netinkamų maršrutų (miestas, kurio nenorite aplankyti, įvedamas klaviatūra) pašalinimą iš sąrašo.

3.2. Grafinės vartotojo sąsajos schema

MARŠRUTŲ PAIEŠKOS

Pradiniai duomenys bei rezultatai apačioje

Įveskite pradinį miestą:
Kaunas

Įveskite didžiausią leistiną gyventojų kiekį mieste:
1000000

Įveskite minimalų maršruto atstumą kilometrais:
100

Duomenų failai:
Choose Files No file chosen

Surasti Atsisiųsti rezultatus

Miestas, kurio nenorite aplankyti:
Ištrinti

Įkeliami abu duomenų dokumentai

Atsisiunčiamas rezultatų dokumentas

Surandami reikiami maršrutai

Norimo ištrinti miesto sąsaja. Įvedamas norimas ištrinti miestas ir paspaudžiamas mygtukas

Paspaudus pavadinimą puslapis grąžinamas į pradinę padėtį

Pradiniai duomenys (maršrutai)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Kaunas	Vilnius	110
Kaunas	Panevėžys	108
Kaunas	Klaipėda	214
Kaunas	Švenčionys	182
Kaunas	Kėdainiai	56
Vilnius	Klaipėda	286
Marijampolė	Klaipėda	286
Šiauliai	Plungė	102

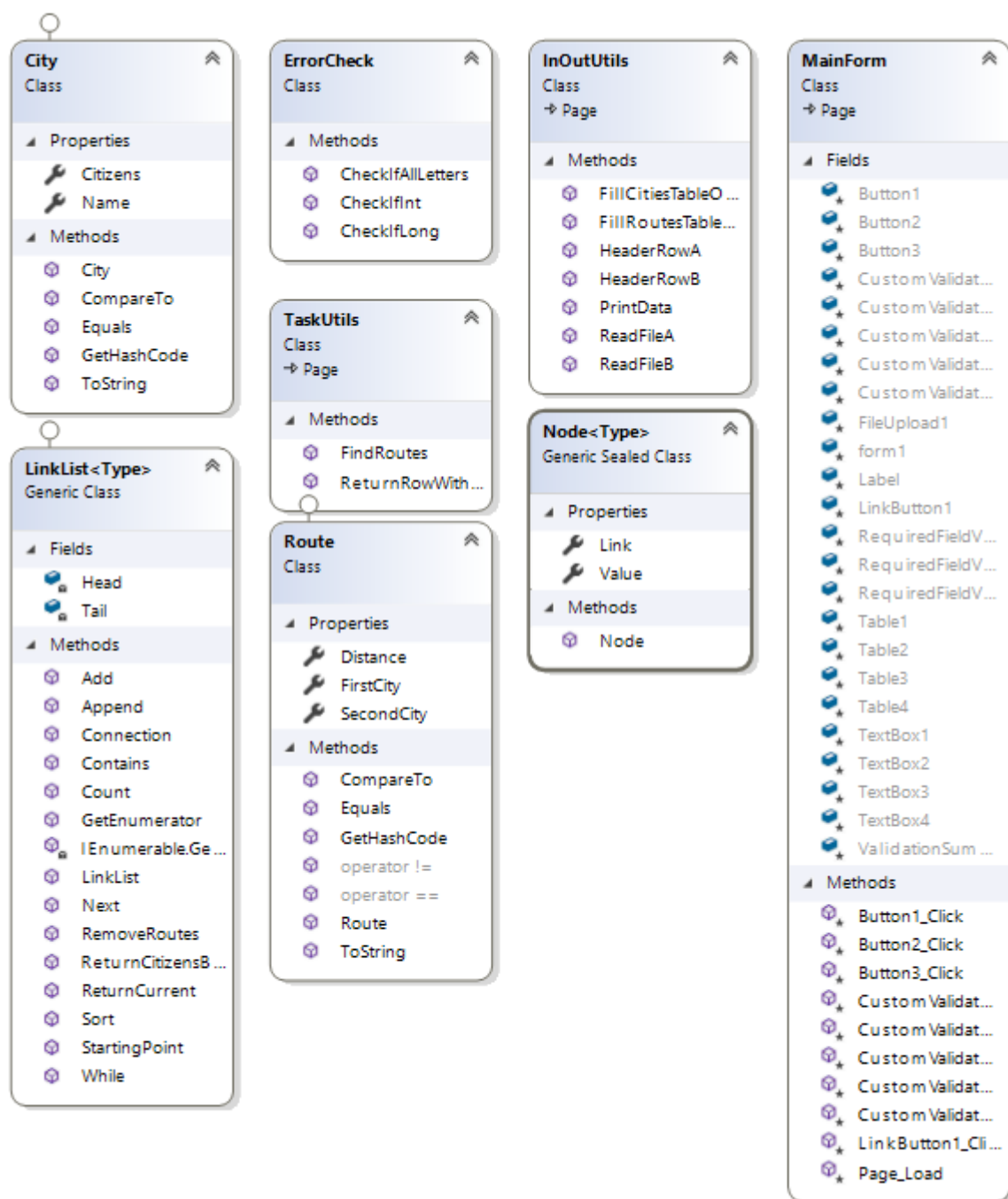
Pradiniai duomenys (miestas)	
Miesto pavadinimas	Gyventojų kiekis
Kaunas	295269
Vilnius	544386
Panevėžys	85885
Klaipėda	152818
Kėdainiai	22677
Švenčionys	4065
Marijampolė	34975
Šiauliai	101511
Plungė	23246

Rezultatai

3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
#TextBox1, #TextBox2, #TextBox3, #TextBox4	Height	35px
#TextBox1, #TextBox2, #TextBox3, #TextBox4	Width	300px
#TextBox4, #Button2, #Button3, #Label, #Table1, #Table2, #Table3, #Table4	Visible	false
#TextBox1, #TextBox2, #TextBox3, #TextBox4	CssClass	margin-bottom: 17px;
#Label	CssClass	margin-top: 40px;
#header	CssClass	Margin-top:30px;
#ValidationSummary1	CssClass	alert alert-danger
#ValidationSummary1	ForeColor	Red
#Button1, #Button2, #Button3	CssClass	btn btn-outline-light
#FileUpload1	AllowMultiple	True
#FileUpload1	class	padding-upload

3.4. Klasų diagrama



3.5. Programos vartotojo vadovas

- ✓ Atsidarius puslapiui, vartotoją pasitinką kelios įvestys bei mygtukas „Surasti“.
- ✓ Norint, jog programa pradėtų veikti, reikia įvesti:
 - Miestą, nuo kurio prasidės visas maršrutas;
 - Didžiausią leistiną gyventojų skaičių mieste, kuris bus maršruto dalimi;
 - Minimalų maršruto atstumą.
- ✓ Norint suformuoti pradinis duomenis, reikia 2 pradinių duomenų dokumentų – „U8a.txt“ bei „U8b.txt“. „U8a.txt“ duomenys išdėstomi tokia tvarka: pirma rašomas pirmojo miesto pavadinimas, antru numeriu – antrojo miesto, o trečia įvestis eilutėje – atstumas tarp šių dviejų miestų. Duomenys yra atskiriami kabliataškiais („;“). Tokia seka, pasirinkus naują eilutę yra taip pat tęsiama.
„U8b.txt“ duomenyse yra aprašomi anksčiau minėto dokumento miestų gyventojų skaičiai. Duomenys išdėstyti tokia tvarka – pirma eina miesto pavadinimas, po to – gyventojų kiekis tame mieste. Viskas taip pat yra skiriama kabliataškiu. Abu dokumentai yra įkeliami po „Duomenų failai:“ antrašte.
- ✓ Paspaudus mygtuką „Surasti“, atsiras trys lentelės. Pirmoje bus pavaizduota pradiniai maršrutų duomenys, antroje – miestų, o trečioje – jau atrinkti maršrutai vartotojui.
- ✓ Taip pat, atsiranda dar vienas naujas įvesties laukelis – į jį galima įvesti miestą, kurio programos naudotojas nenorėtų aplankyti. Tas miestas bus pašalintas iš galutinio sąrašo bei lentelės.
- ✓ Paspaudus „Atsisiųsti rezultatus“ mygtuką, galima atsisiųsti išspausdintus rezultatus lentele.

3.6. Programos tekstas

City.cs

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

namespace LD2_WebApp
{
    public class City : IComparable<City>, IEquatable<City>
    {
        public string Name { get; set; }
        public long Citizens { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="city">City's name</param>
        /// <param name="citizens">Amount of citizens</param>
        public City(string city, long citizens)
        {
            this.Name = city;
            this.Citizens = citizens;
        }

        // Compares two objects by default
        public int CompareTo(City other)
        {
            return Citizens.CompareTo(other.Citizens);
        }
    }
}
```

```

    }
    //Equals override
    public bool Equals(City other)
    {
        return Name == other.Name && Citizens == other.Citizens;
    }

    //GetHashCode() override
    public override int GetHashCode()
    {
        return Name.GetHashCode() ^ Citizens.GetHashCode();
    }

    //ToString() override
    public override string ToString()
    {
        string line = String.Format("|{0, -20}|{1, 20}|", Name, Citizens);
        return line;
    }
}
}

```

Route.cs

```

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

namespace LD2_WebApp
{
    public class Route : IComparable<Route>, IEquatable<Route>
    {
        public string FirstCity { get; set; }
        public string SecondCity { get; set; }
        public int Distance { get; set; }

        public Route(string firstCity, string secondCity, int distance)
        {
            this.FirstCity = firstCity;
            this.SecondCity = secondCity;
            this.Distance = distance;
        }

        /// <summary>
        /// Compares by distance and first city's name
        /// </summary>
        /// <param name="route">Route object to compare to</param>
        /// <returns></returns>
        public int CompareTo(Route route)
        {
            if (this.Distance == route.Distance)
            {
                return this.FirstCity.CompareTo(route.FirstCity);
            }

            else
            {
                return this.Distance.CompareTo(route.Distance);
            }
        }

        /// <summary>
        /// Equals method override

```



```

    /// </summary>
    /// <param name="other">object to compare to</param>
    /// <returns>a true or false statement</returns>
    public bool Equals(Route other)
    {
        return FirstCity == other.FirstCity && SecondCity == other.SecondCity && Distance
== other.Distance;
    }

    //GetHashCode() method override
    public override int GetHashCode()
    {
        return FirstCity.GetHashCode() ^ SecondCity.GetHashCode() ^
Distance.GetHashCode();
    }

    /// <summary>
    /// Equals operator
    /// </summary>
    /// <param name="a">First Route object</param>
    /// <param name="b">Second route object to compare with</param>
    /// <returns>a true or false statement</returns>
    public static bool operator == (Route a, Route b)
    {
        return a.FirstCity == b.FirstCity && a.SecondCity == b.SecondCity && a.Distance ==
b.Distance;
    }

    /// <summary>
    /// Not equals operator
    /// </summary>
    /// <param name="a">First Route object</param>
    /// <param name="b">Second route object to compare with</param>
    /// <returns>a true or false statement</returns>

    public static bool operator != (Route a, Route b)
    {
        return a.FirstCity != b.FirstCity && a.SecondCity != b.SecondCity && a.Distance !=
b.Distance;
    }

    //ToString() method override
    public override string ToString()
    {
        string line;
        return line = String.Format("|{0, -20}|{1, -20}|{2, 15}|", FirstCity, SecondCity,
Distance);
    }
}

```

Node.cs

```

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

namespace LD2_WebApp
{
    sealed class Node<Type>
    {
        public Type Value { get; set; }
    }
}

```

```

        public Node<Type> Link { get; set; }
        public Node(Type value, Node<Type> link)
        {
            this.Value = value;
            this.Link = link;
        }
    }
}

```

LinkedList.cs

```

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

namespace LD2_WebApp
{
    public class LinkedList<Type> : IEnumerable<Type> where Type : IComparable<Type>,
    IEquatable<Type>
    {
        private Node<Type> Head;
        private Node<Type> Tail;

        /// <summary>
        /// Constructor
        /// </summary>
        public LinkedList()
        {
            this.Head = null;
            this.Tail = null;
        }

        /// <summary>
        /// Adds a new object to the link list
        /// </summary>
        /// <param name="value">object to add</param>
        public void Add(Type value)
        {
            Node<Type> newNode = new Node<Type>(value, null);
            if (Head == null)
            {
                Head = newNode;
                Tail = newNode;
            }

            else
            {
                Tail.Link = newNode;
                Tail = newNode;
            }
        }

        /// <summary>
        /// Sets the starting point to link list's head
        /// </summary>
        public void StartingPoint() => this.Tail = this.Head;

        /// <summary>
        /// Moves the list's tail forward across the references
        /// </summary>
        public void Next()
        {

```

```

        this.Tail = this.Tail.Link;
    }

    /// <summary>
    /// Checks if the tail is equal to null
    /// </summary>
    /// <returns>true if tail is not equal to null and vice versa</returns>
    public bool While()
    {
        return this.Tail != null;
    }

    /// <summary>
    /// Returns current tail object (if the tail equals null, returns a null)
    /// </summary>
    /// <returns>object's value or null</returns>
    public Type ReturnCurrent()
    {
        if (Tail == null)
        {
            return default(Type);
        }

        return Tail.Value;
    }

    /// <summary>
    /// Returns the amount of citizens by the city's name
    /// </summary>
    /// <param name="cityName">name of the city</param>
    /// <returns>amount of citizens</returns>
    public long ReturnCitizensByName(string cityName)
    {
        if (!(this is LinkList<City>))
        {
            return -1;
        }

        for (Node<Type> w = Head; w != null; w = w.Link)
        {
            var obj = w.Value as City;
            if (obj.Name == cityName) return obj.Citizens;
        }
        return -1;
    }

    /// <summary>
    /// Sorts the link list by a specific order
    /// </summary>
    public void Sort()
    {
        if (!(this is LinkList<Route>)) return;
        Node<Type> a = Head;
        while (a != null)
        {
            Node<Type> b = a;
            Node<Type> min = a;
            while (b != null)
            {
                if (min.Value.CompareTo(b.Value) > 0)
                {
                    min = b;
                }
                b = b.Link;
            }
        }
    }

```

```

        var temp= a.Value;
        a.Value = min.Value;
        min.Value = temp;
        a = a.Link;
    }
}

/// <summary>
/// Removes Route objects from the list
/// </summary>
/// <param name="cityName">name of one of the cities in Route object to be removed
</param>
public void RemoveRoutes(string cityName)
{
    if (!(this is LinkedList<Route>)) return;
    Node<Type> current = Head;
    while (current != null)
    {
        Node<Route> route = current as Node<Route>;
        if ((route.Value.FirstCity == cityName || route.Value.SecondCity == cityName)
&& current == Head)
        {
            Head = Head.Link;
        }

        else if ((route.Value.FirstCity == cityName || route.Value.SecondCity ==
cityName))
        {
            Node<Type> j;
            for (j = Head; j.Link != current; j = j.Link) ;
            j.Link = current.Link;
        }

        current = current.Link;
    }
}

/// <summary>
/// Checks if the list contains a specific object
/// </summary>
/// <param name="type"></param>
/// <returns></returns>
public bool Contains(Type type)
{
    for (Node<Type> temp = Head; temp != null; temp = temp.Link)
    {
        if (this is LinkedList<Route> && temp.Value as Route == (type as Route))
        {
            return true;
        }

        if (this is LinkedList<City> && temp.Value as City == (type as City))
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Finds a connection between Route object and other routes
/// </summary>
/// <param name="startingCity">name of the user inputted starting city</param>
/// <param name="route">Route object to compare with</param>

```

```

    /// <returns> if the Route object fits the criteria returns true and vice
    versa</returns>
    public bool Connection(string startingCity, Route route)
    {
        if (!(this is LinkList<Route>)) return false;
        for (Node<Type> w = Head; w != null; w = w.Link)
        {
            var temp = w.Value as Route;
            if (temp.FirstCity == startingCity && temp.SecondCity == route.FirstCity)
            {
                return true;
            }
        }
        return false;
    }

    /// <summary>
    /// Returns the amount of objects in the list
    /// </summary>
    /// <returns> amount of list's objects</returns>
    public int Count()
    {
        int count = 0;
        for (Node<Type> a = Head; a != null; a = a.Link)
        {
            count++;
        }
        return count;
    }

    /// <summary>
    /// Adds link's objects to the string array
    /// </summary>
    /// <param name="AllLines">name of the string array</param>
    /// <param name="index">index to know which line is not taken</param>
    public void Append(string[] AllLines, ref int index)
    {
        if (Count() == 0 && this is LinkList<Route>)
        {
            AllLines[index++] = "Tokių maršrutų nėra.";
            return;
        }

        else if (Count() == 0 && this is LinkList<City>)
        {
            AllLines[index++] = "Miestų nėra.";
            return;
        }

        if (this is LinkList<Route>)
        {
            AllLines[index++] = String.Format("|{0, -20}|{1, -20}|{2, 15}|", "Pirmas
miestas", "Antras miestas", "Atstumas");
        }

        if (this is LinkList<City>)
        {
            AllLines[index++] = String.Format("|{0, -20}|{1, 20}|", "Miesto pavadinimas",
"Gyventojų kiekis");
        }

        for (Node<Type> w = Head; w != null; w = w.Link)
        {
            AllLines[index++] = w.Value.ToString();
        }
    }

```

```

    /// <summary>
    /// GetEnumerator() (generic) method fill
    /// </summary>
    /// <returns>value of objects one by one</returns>
    public IEnumerator<Type> GetEnumerator()
    {
        for (Node<Type> w = Head; w != null; w = w.Link)
        {
            yield return w.Value;
        }
    }

    /// <summary>
    /// GetEnumerator() (generic) method fill
    /// </summary>
    /// <returns>value of objects one by one</returns>
    IEnumerator IEnumerable.GetEnumerator()
    {
        for (Node<Type> w = Head; w != null; w = w.Link)
        {
            yield return w.Value;
        }
    }
}

```

```

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

namespace LD2_WebApp
{
    class InOutUtils : System.Web.UI.Page
    {
        public static LinkList<Route> ReadFileA(StreamReader wholeFile)
        {
            LinkList<Route> AllRoutes = new LinkList<Route>();
            using (wholeFile)
            {
                for (string line = wholeFile.ReadLine(); line != null; line =
wholeFile.ReadLine())
                {
                    string[] AllParts = line.Split(';');
                    string cityA = AllParts[0];
                    string cityB = AllParts[1];
                    int distance = int.Parse(AllParts[2]);
                    Route route = new Route(cityA, cityB, distance);
                    AllRoutes.Add(route);
                }
            }
            return AllRoutes;
        }

        public static LinkList<City> ReadFileB(StreamReader wholeFile)
        {
            LinkList<City> AllCities = new LinkList<City>();
            string line;
            using (wholeFile)
            {
                while ((line = wholeFile.ReadLine()) != null)
                {
                    string[] AllParts = line.Split(';');
                    string name = AllParts[0];
                    long citizens = long.Parse(AllParts[1]);
                    City city = new City(name, citizens);
                    AllCities.Add(city);
                }
            }
            return AllCities;
        }

        /// <summary>
        /// Fills RouteLList table on screen
        /// </summary>
        /// <param name="table">table to modify</param>
        /// <param name="filler">list to take data from</param>
        public static void FillRoutesTableOnScreen(Table table, LinkList<Route> filler)
        {
            if (filler == null || filler.Count() == 0)
            {
                TableRow row = new TableRow();
                row.Cells.Add(new TableCell { ColumnSpan = 3, Text = "Maršrutų nėra." });
                table.Rows.Add(row);
            }
        }
    }
}

```

```

        return;
    }
    table.Rows.Add(HeaderRowA());
    foreach (Route route in filler)
    {
        TableRow row = new TableRow();
        row.Cells.Add(new TableCell { Text = route.FirstCity });
        row.Cells.Add(new TableCell { Text = route.SecondCity });
        row.Cells.Add(new TableCell { Text = (route.Distance).ToString() });
        table.Rows.Add(row);
    }
}

/// <summary>
/// Fills CityLList table on screen
/// </summary>
/// <param name="table">table to modify</param>
/// <param name="filler">CityLList object to take data from</param>
public static void FillCitiesTableOnScreen(Table table, LinkList<City> filler)
{
    if (filler.Count() == 0)
    {
        TableRow row = new TableRow();
        row.Cells.Add(new TableCell { ColumnSpan = 3, Text = "Miestų nėra." });
        table.Rows.Add(row);
        return;
    }
    table.Rows.Add(HeaderRowB());
    foreach (City city in filler)
    {
        TableRow row = new TableRow();
        row.Cells.Add(new TableCell { Text = city.Name });
        row.Cells.Add(new TableCell { Text = (city.Citizens).ToString() });
        table.Rows.Add(row);
    }
}

/// <summary>
/// Header row for Route class objects
/// </summary>
/// <returns>a header row for the table</returns>
public static TableRow HeaderRowA()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell { Text = "Pirmas miestas" });
    row.Cells.Add(new TableCell { Text = "Antras miestas" });
    row.Cells.Add(new TableCell { Text = "Atstumas tarp miestų" });
    return row;
}

/// <summary>
/// Header row for City class objects
/// </summary>
/// <returns>a header row for the table</returns>
public static TableRow HeaderRowB()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell { Text = "Miesto pavadinimas" });
    row.Cells.Add(new TableCell { Text = "Gyventojų kiekis" });
    return row;
}

/// <summary>
/// Combines all the data and "prints" it into a string array
/// </summary>
/// <param name="start1">list of all the starting data (Route objects)</param>

```



```

    /// <param name="start2">list of all the starting data (City objects)</param>
    /// <param name="end">list of filtered Route class objects</param>
    /// <returns>a string array</returns>
    public static string[] PrintData(LinkList<Route> start1, LinkList<City> start2,
LinkList<Route> end)
    {
        string[] AllLines = new string[start1.Count() + start2.Count() + end.Count() + 8];
        int index = 0;
        AllLines[index++] = String.Format("Pradiniai duomenys");
        start1.Append(AllLines, ref index);
        AllLines[index++] = String.Empty;
        start2.Append(AllLines, ref index);
        AllLines[index++] = String.Empty;
        AllLines[index++] = String.Format("Rezultatai");
        end.Append(AllLines, ref index);

        return AllLines;
    }
}

```

TaskUtils.cs

```

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

namespace LD2_WebApp
{
    class TaskUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Finds all routes that fit the criteria
        /// </summary>
        /// <param name="startingCity">name of the starting city(chosen by user)</param>
        /// <param name="maxCitizens">maximum amount of citizens allowed in a city (chosen by
user)</param>
        /// <param name="minDistance">minimum route distance</param>
        /// <param name="AllRoutes">All routes from starting file</param>
        /// <param name="AllCities">All cities from starting file</param>
        /// <returns></returns>
        public static LinkList<Route> FindRoutes(string startingCity, long maxCitizens, int
minDistance, LinkList<Route> AllRoutes, LinkList<City> AllCities)
        {
            LinkList<Route> possibleRoutes = new LinkList<Route>();
            foreach (Route route in AllRoutes)
            {
                if (((route.FirstCity == startingCity && route.Distance >= minDistance) ||
(route.FirstCity != startingCity && AllRoutes.Connection(startingCity, route) &&
route.Distance >= minDistance && AllCities.ReturnCitizensByName(route.FirstCity) <=
maxCitizens
&& AllCities.ReturnCitizensByName(route.SecondCity) <= maxCitizens)) &&
!possibleRoutes.Contains(route))
                {
                    possibleRoutes.Add(route);
                }
            }
            return possibleRoutes;
        }
    }
}

```

```

    /// <summary>
    /// Returns a row with specific test
    /// </summary>
    /// <param name="text">text to put in a row</param>
    /// <param name="n">amount of cell's column span</param>
    /// <returns></returns>
    public static TableRow ReturnRowWithText(string text, int n)
    {
        TableRow row = new TableRow();
        row.Cells.Add(new TableCell { Text = text, ColumnSpan = n });
        return row;
    }
}

```

ErrorCheck.cs

```

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

namespace LD2_WebApp
{
    public class ErrorCheck
    {
        /// <summary>
        /// Checks if a given string parses to long or not
        /// </summary>
        /// <param name="value">string to parse from</param>
        /// <returns>a true or false statement</returns>
        public static bool CheckIfLong(string value)
        {
            bool c = long.TryParse(value, out long number);
            return c;
        }

        /// <summary>
        /// Checks if a given string parses into integer
        /// </summary>
        /// <param name="value">name of the string to parse from</param>
        /// <returns>a true or false statement</returns>
        public static bool CheckIfInt(string value)
        {
            bool c = int.TryParse(value, out int number);
            return c;
        }

        /// <summary>
        /// Checks if the string contains only letters and no special symbols
        /// </summary>
        /// <param name="value">string input to check</param>
        /// <returns>true or false statement based on letters count in string compared to
        string's length</returns>
        public static bool CheckIfAllLetters(string value)
        {
            int count = 0;
            for (int i = 0; i < value.Count(); i++)
            {
                if (char.IsLetter(value[i]) || (value[i] == ' ') && i != 0)
                {
                    count++;
                }
            }
            if (count == value.Count())

```

```

        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
}

```

LinkedListTests.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xunit;
using LD2_WebApp;

namespace LD2_WebAppTests
{
    //Testing with Route class
    public class LinkedListTests
    {
        LinkedList<Route> Routes = new LinkedList<Route>();

        [Fact]
        public void Should_Be_Null()
        {
            Routes = new LinkedList<Route>();

            Assert.Null(Routes.ReturnCurrent());
        }

        [Fact]
        public void Should_Add_To_Beginning()
        {
            Routes = new LinkedList<Route>();

            Routes.Add(new Route("Kaunas", "Panevėžys", 113));

            Route expected = new Route("Kaunas", "Panevėžys", 113);

            Assert.Equal(Routes.ReturnCurrent(), expected);
        }

        [Fact]
        public void Should_Add_New_Objects()
        {
            int expected = 3;

            Routes = new LinkedList<Route>();

            Route route1 = new Route("Kaunas", "Vilnius", 100);
            Route route2 = new Route("Kaunas", "Panevėžys", 113);
            Route route3 = new Route("Kaunas", "Klaipėda", 150);

            Routes.Add(route1);
            Routes.Add(route2);
            Routes.Add(route3);

            Assert.Equal(Routes.Count(), expected);
        }
    }
}

```

```

[Fact]
public void StartingPoint_Should_Be_Head()
{
    Route expected = new Route("Kaunas", "Panevėžys", 113);

    Routes = new LinkList<Route>();

    Routes.Add(expected);
    Routes.Add(new Route("", "", 0));

    Routes.StartingPoint();

    Assert.Equal(Routes.ReturnCurrent(), expected);
}

[Fact]
public void Next_Should_Point_Forward()
{
    Route expected = new Route("Kaunas", "Panevėžys", 113);

    Routes = new LinkList<Route>();

    Routes.Add(new Route("", "", 0));
    Routes.Add(expected);

    Routes.StartingPoint();
    Routes.Next();

    Assert.Equal(Routes.ReturnCurrent(), expected);
}

[Fact]
public void While_Should_Return_False()
{
    bool expected = false;

    Routes = new LinkList<Route>();

    bool result = Routes.While();

    Assert.Equal(result, expected);
}

[Fact]
public void Should_Return_Current_Latest_Object()
{
    Route expected = new Route("", "", 0);

    Routes = new LinkList<Route>();

    Routes.Add(expected);

    Assert.Equal(Routes.ReturnCurrent(), expected);
}

[Theory]
[InlineData(100000)]
[InlineData(100)]
[InlineData(9764645)]
public void Should_Return_Citizens_By_Object_Type(long expected)
{
    LinkList<City> Cities = new LinkList<City>();

    Cities.Add(new City("Test1", 100000));
    Cities.Add(new City("Test2", 100000));
    Cities.Add(new City("Test3", 50000));
}

```

```

        Cities.Add(new City("Test", expected));
        Cities.Add(new City("Test4", 600000));

        long result = Cities.ReturnCitizensByName("Test");

        Assert.Equal(result, expected);
    }

    [Fact]
    public void Should_Return_Negative_One()
    {
        int expected = -1;

        Routes = new LinkList<Route>();

        Assert.Equal(Routes.ReturnCitizensByName(""), expected);
    }

    [Fact]
    public void Should_Sort_Correctly_By_Distance()
    {
        Route expected = new Route("A", "", 5);
        Routes = new LinkList<Route>();

        Routes.Add(new Route("F", "", 1000));
        Routes.Add(new Route("C", "", 100));
        Routes.Add(expected);
        Routes.Add(new Route("A", "", 70));

        Routes.Sort();

        Routes.StartingPoint();

        Assert.Equal(Routes.ReturnCurrent(), expected);
    }

    [Fact]
    public void Should_Sort_Correctly_By_FirstCity()
    {
        Route expected = new Route("A", "", 70);
        Routes = new LinkList<Route>();

        Routes.Add(new Route("F", "", 1000));
        Routes.Add(new Route("B", "", 70));
        Routes.Add(new Route("C", "", 100));
        Routes.Add(expected);

        Routes.Sort();

        Routes.StartingPoint();

        Assert.Equal(Routes.ReturnCurrent(), expected);
    }

    [Fact]
    public void Should_RemoveRoutes()
    {
        string indicator = "A";

        Routes = new LinkList<Route>();

        Routes.Add(new Route("B", "", 100000));
        Routes.Add(new Route(indicator, "", 100000));
        Routes.Add(new Route("C", "", 100000));
        Routes.Add(new Route("", indicator, 100000));
    }

```

```

Routes.RemoveRoutes(indicator);

bool checker = true, results = false;

foreach (Route route in Routes)
{
    if (route.FirstCity == indicator || route.SecondCity == indicator)
    {
        checker = false;
    }
}

if (checker && Routes.Count() == 2)
{
    results = true;
}

Assert.True(results);
}

[Fact]
public void Should_Remove_First_Object()
{
    Routes = new LinkedList<Route>();

    Routes.Add(new Route("A", "", 100000));

    Routes.RemoveRoutes("A");

    Assert.False(Routes.Contains(new Route("A", "", 100000)));
}

[Fact]
public void Should_Remove_Last_Object()
{
    Routes = new LinkedList<Route>();

    Routes.Add(new Route("A", "", 100000));
    Routes.Add(new Route("B", "", 100000));
    Routes.Add(new Route("C", "", 100000));
    Routes.Add(new Route("D", "", 100000));

    Routes.RemoveRoutes("D");

    Assert.False(Routes.Contains(new Route("D", "", 100000)));
}

[Fact]
public void Should_Contain_Object()
{
    Routes = new LinkedList<Route>();

    Route duplicated = new Route("Duplicated", "B", 12121);

    Routes.Add(new Route("", "", 10000));
    Routes.Add(duplicated);
    Routes.Add(new Route("ABC", "", 1334545));
    Routes.Add(new Route("DEF", "", 67942));

    Assert.True(Routes.Contains(duplicated));
}

[Fact]
public void Should_Find_A_Connection()
{
    string startingCity = "Kaunas";

```

```

        Routes = new LinkedList<Route>();

        Routes.Add(new Route(startingCity, "Vilnius", 12031));
        Route indicator = new Route("Vilnius", "Panevėžys", 1638456);

        Assert.True(Routes.Connection(startingCity, indicator));
    }

    [Fact]
    public void Should_Be_No_Connection()
    {
        string startingCity = "Kaunas";
        Routes = new LinkedList<Route>();

        Routes.Add(new Route("Klaipėda", "Vilnius", 12031));
        Route indicator = new Route("Vilnius", "Panevėžys", 1638456);

        Assert.False(Routes.Connection(startingCity, indicator));
    }

    [Fact]
    public void Should_Count_Correctly()
    {
        int expected = 4;

        Routes = new LinkedList<Route>();
        Routes.Add(new Route("", "", 0));
        Routes.Add(new Route("", "", 0));
        Routes.Add(new Route("", "", 0));
        Routes.Add(new Route("", "", 0));

        Assert.Equal(Routes.Count(), expected);
    }

    [Fact]
    public void Should_Append_Objects()
    {
        bool result = false;

        Routes = new LinkedList<Route>();

        Routes.Add(new Route("", "", 0));
        Routes.Add(new Route("", "", 1));

        string[] ToStringLines = new string[Routes.Count() + 1];

        int ind = 0;
        Routes.Append(ToStringLines, ref ind);

        if (ToStringLines[ind - 2] == (new Route("", "", 0)).ToString() &&
            ToStringLines[ind - 1] == (new Route("", "", 1)).ToString())
        {
            result = true;
        }

        Assert.True(result);
    }

    [Fact]
    public void Should_Append_Error_Message()
    {
        string expected = "Tokių maršrutų nėra.";

        Routes = new LinkedList<Route>();

        string[] ToStringLines = new string[Routes.Count() + 1];

```

```

        int ind = 0;
        Routes.Append(ToStringLines, ref ind);

        Assert.Equal(ToStringLines[0], expected);
    }

    [Fact]
    public void Should_Use_GetEnumerator_Correctly()
    {
        Route expected = new Route("A", "B", 10000);

        Routes = new LinkList<Route>
        {
            expected
        };

        Route result = null;

        foreach (Route route in Routes)
        {
            result = route;
        }

        Assert.Equal(expected, result);
    }
}

```

MainForm.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 LD2_WebApp
{
    public partial class MainForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Page.IsPostBack && Session["TABLE1"] != null && Session["TABLE2"] != null &&
                Session["TABLE3"] != null && Table1.Rows.Count == 0 && Table2.Rows.Count == 0 &&
                Table3.Rows.Count == 0)
            {
                Table1.Rows.AddRange(((TableRow[])Session["TABLE1"])); //Checks if there is
                any data to restore on page after postback
                Table2.Rows.AddRange(((TableRow[])Session["TABLE2"]));
                Table3.Rows.AddRange(((TableRow[])Session["TABLE3"]));
            }
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid) //initiates validation
            {
                const string CFr = "App_Data/Rezultatai.txt";
                Session["CFr"] = CFr;

                if (!FileUpload1.HasFile || (FileUpload1.PostedFiles.Count == 2 &&
                    (!FileUpload1.PostedFiles[0].FileName.EndsWith(".txt")) ||

```



```

!FileUpload1.PostedFiles[1].FileName.EndsWith(".txt"))) || FileUpload1.PostedFiles.Count > 2
|| FileUpload1.PostedFiles.Count < 2)
    { //Checks if the file upload has 2 files
        Session["Files"] = false;
        Page.Validate();
        return;
    }

    StreamReader AllLinesA = new
StreamReader(FileUpload1.PostedFiles[0].InputStream); //reads starting data
    StreamReader AllLinesB = new
StreamReader(FileUpload1.PostedFiles[1].InputStream);

    LinkedList<Route> AllRoutes = InOutUtils.ReadFileA(AllLinesA); //puts starting
data into lists
    LinkedList<City> AllCities = InOutUtils.ReadFileB(AllLinesB);

    string startingCity = TextBox1.Text;
    long maxCitizens = long.Parse(TextBox2.Text);
    int minDistance = int.Parse(TextBox3.Text);

    LinkedList<Route> FilteredRoutes = TaskUtils.FindRoutes(startingCity,
maxCitizens, minDistance, AllRoutes, AllCities); //filters routes
    FilteredRoutes.Sort();

    Table1.Rows.Clear(); //clears tables rows to prevent data duplication
    Table2.Rows.Clear();
    Table3.Rows.Clear();

    Table1.Rows.Add(TaskUtils.ReturnRowWithText("Pradiniai duomenys (maršrutai)",
3));
    InOutUtils.FillRoutesTableOnScreen(Table1, AllRoutes); //fills first table

    Table2.Rows.Add(TaskUtils.ReturnRowWithText("Pradiniai duomenys (miestai)",
2));
    InOutUtils.FillCitiesTableOnScreen(Table2, AllCities); //fills second table

    Table3.Rows.Add(TaskUtils.ReturnRowWithText("Rezultatai", 3));
    InOutUtils.FillRoutesTableOnScreen(Table3, FilteredRoutes); //fills third
table

    Table1.Visible = true; //makes hidden tables visible
    Table2.Visible = true;
    Table3.Visible = true;

    string[] AllLines = InOutUtils.PrintData(AllRoutes, AllCities,
FilteredRoutes);

    if (File.Exists(Server.MapPath(CFr))) //checks if the result file exists to
prevent data duplication
    {
        File.Delete(Server.MapPath(CFr));
    }

    File.AppendAllLines(Server.MapPath(CFr), AllLines); //appends results

    Button2.Visible = true; //makes hidden controls visible
    Button3.Visible = true;
    TextBox4.Visible = true;
    Label1.Visible = true;

    TableRow[] rows1 = new TableRow[Table1.Rows.Count]; //prepares for data
preservation
    Table1.Rows.CopyTo(rows1, 0);
    Session.Remove("TABLE1"); //prevents data duplication
    Session.Add("TABLE1", rows1); //preserves data

```

```

        TableRow[] rows2 = new TableRow[Table2.Rows.Count];
        Table2.Rows.CopyTo(rows2, 0);
        Session.Remove("TABLE2");
        Session.Add("TABLE2", rows2);

        TableRow[] rows3 = new TableRow[Table3.Rows.Count];
        Table3.Rows.CopyTo(rows3, 0);
        Session.Remove("TABLE3");
        Session.Add("TABLE3", rows3);

        Session["RouteList"] = FilteredRoutes; //preserves filtered routes list
    }
}

protected void Button2_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string CFr = (string)Session["CFr"];
        string removeCity = TextBox4.Text; //finds out what city to remove
        LinkList<Route> filtered = (LinkList<Route>)Session["RouteList"]; //gets
filtered route from Session
        filtered.RemoveRoutes(removeCity); //removes chosen city from result's list
        Table4.Rows.Add(TaskUtils.ReturnRowWithText("Rezultatai (po panaikinimo)",
3));
        InOutUtils.FillRoutesTableOnScreen(Table4, filtered);
        Table4.Visible = true; //reveals another hidden table

        string[] AllLines = new string[filtered.Count() + 3];
        int index = 0;
        AllLines[index++] = "Rezultatai (po panaikinimo)";
        filtered.Append(AllLines, ref index);
        File.AppendAllLines(Server.MapPath(CFr), AllLines); //fills results file
    }
}

protected void Button3_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string CFr = (string)Session["CFr"];
        //downloads results file
        FileStream results = File.OpenRead(Server.MapPath("App_Data/" + CFr.Remove(0,
9))); //reads file
        byte[] temp = new byte[results.Length];
        results.Read(temp, 0, Convert.ToInt32(results.Length)); //converts
        results.Close();
        Response.AddHeader("Content-disposition", "attachment; filename=" +
CFr.Remove(0, 9)); //sets file's properties
        Response.ContentType = "application/octet-stream";
        Response.BinaryWrite(temp);
        Response.End(); //sends file to download
    }
}

protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!ErrorCheck.CheckIfLong(TextBox2.Text)) //checks if the inputted string is long
    {
        args.IsValid = false;
    }
}

```

```

        else
        {
            args.IsValid = true;
        }
    }

protected void CustomValidator2_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!ErrorCheck.CheckIfInt(TextBox3.Text)) //checks if the inputed string is int
    {
        args.IsValid = false;
    }

    else
    {
        args.IsValid = true;
    }
}

protected void CustomValidator3_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!ErrorCheck.CheckIfAllLetters(TextBox1.Text)) //checks if the inputed string
is a word or a combination of words
    {
        args.IsValid = false;
    }

    else
    {
        args.IsValid = true;
    }
}

protected void CustomValidator4_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!ErrorCheck.CheckIfAllLetters(TextBox4.Text))
    {
        args.IsValid = false;
    }

    else
    {
        args.IsValid = true;
    }
}

protected void CustomValidator5_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (Session["Files"] != null && !(bool)Session["Files"])
    {
        args.IsValid = false;
        Session["Files"] = true;
    }

    else
    {
        args.IsValid = true;
    }
}

protected void LinkButton1_Click(object sender, EventArgs e)
{

```

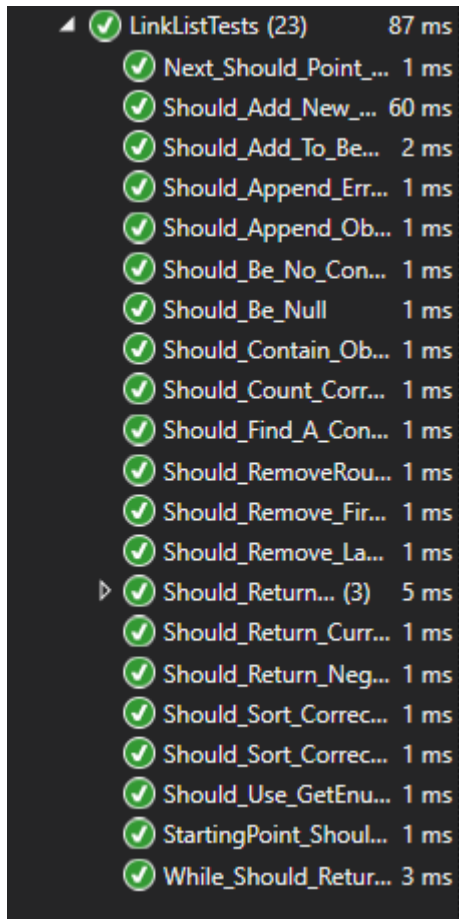
```

        Page.Response.Redirect(Page.Request.RawUrl);
    }
}

```

3.7. Pradiniai duomenys ir rezultatai

Testai



LinkListTests (23)	87 ms
Next_Should_Point...	1 ms
Should_Add_New...	60 ms
Should_Add_To_Be...	2 ms
Should_Append_Err...	1 ms
Should_Append_Ob...	1 ms
Should_Be_No_Con...	1 ms
Should_Be_Null	1 ms
Should_Contain_Ob...	1 ms
Should_Count_Corr...	1 ms
Should_Find_A_Con...	1 ms
Should_RemoveRou...	1 ms
Should_Remove_Fir...	1 ms
Should_Remove_La...	1 ms
Should_Return... (3)	5 ms
Should_Return_Curr...	1 ms
Should_Return_Neg...	1 ms
Should_Sort_Correc...	1 ms
Should_Sort_Correc...	1 ms
Should_Use_GetEnu...	1 ms
StartingPoint_Shoul...	1 ms
While_Should_Retur...	3 ms

1) variantas

U8a.txt

Kaunas;Vilnius;110

Kaunas;Panevėžys;108

Kaunas;Klaipėda;214

Kaunas;Švenčionys;182

Kaunas;Kėdainiai;56

Vilnius;Klaipėda;286

Marijampolė;Klaipėda;286

Šiauliai;Plungė;102

U8b.txt

Kaunas;295269

Vilnius;544386

Panevėžys;85885

Klaipėda;152818

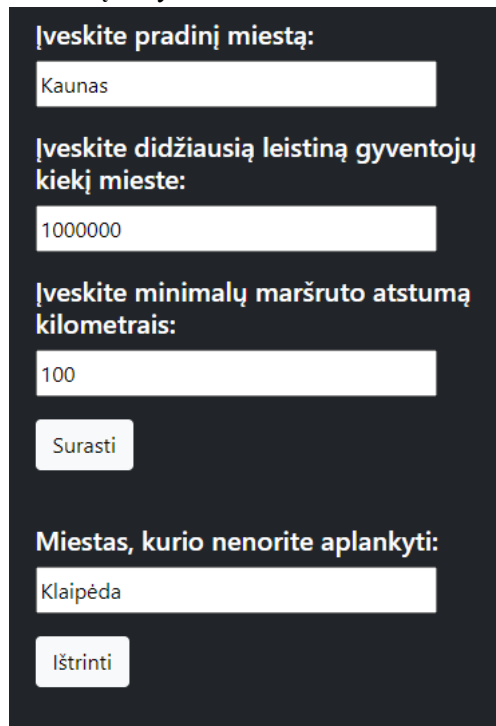
Kėdainiai;22677

Švenčionys;4065

Marijampolė;34975

Šiauliai;101511
Plungė;23246

Ekrano įvestys:



The screenshot shows a dark-themed web form with white text and input fields. The form contains the following elements:

- A label "Įveskite pradinį miestą:" followed by a text input field containing "Kaunas".
- A label "Įveskite didžiausią leistiną gyventojų kiekį mieste:" followed by a text input field containing "1000000".
- A label "Įveskite minimalų maršruto atstumą kilometrais:" followed by a text input field containing "100".
- A button labeled "Surasti".
- A label "Miestas, kurio nenorite aplankyti:" followed by a text input field containing "Klaipėda".
- A button labeled "Ištrinti".

Rezultatai.txt

Rezultatai - Notepad		
File Edit Format View Help		
Pradiniai duomenys		
Pirmas miestas	Antras miestas	Atstumas
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Kaunas	Kėdainiai	56
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Miesto pavadinimas	Gyventojų kiekis	
Kaunas	295269	
Vilnius	544386	
Panevėžys	85885	
Klaipėda	152818	
Kėdainiai	22677	
Švenčionys	4065	
Marijampolė	34975	
Šiauliai	101511	
Plungė	23246	
Rezultatai		
Pirmas miestas	Antras miestas	Atstumas
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Vilnius	Klaipėda	286
Rezultatai (po panaikinimo)		
Pirmas miestas	Antras miestas	Atstumas
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182

Pradiniai duomenys (maršrutai)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Kaunas	Vilnius	110
Kaunas	Panevėžys	108
Kaunas	Klaipėda	214
Kaunas	Švenčionys	182
Kaunas	Kėdainiai	56
Vilnius	Klaipėda	286
Marijampolė	Klaipėda	286
Šiauliai	Plungė	102

Pradiniai duomenys (miestai)	
Miesto pavadinimas	Gyventojų kiekis
Kaunas	295269
Vilnius	544386
Panevėžys	85885
Klaipėda	152818
Kėdainiai	22677
Švenčionys	4065
Marijampolė	34975
Šiauliai	101511
Plungė	23246

Rezultatai		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Kaunas	Panevėžys	108
Kaunas	Vilnius	110
Kaunas	Švenčionys	182
Kaunas	Klaipėda	214
Vilnius	Klaipėda	286

2 variantas)

U8a.txt

Vilnius;Klaipėda;286

Marijampolė;Klaipėda;286

Šiauliai;Plungė;102

U8b.txt

Vilnius;100000

Klaipėda;1552233

Marijampolė;15522

Šiauliai;1545452

Plungė;155222

Rezultatai.txt

Rezultatai - Notepad		
File Edit Format View Help		
Pradiniai duomenys		
Pirmas miestas	Antras miestas	Atstumas
Šiauliai	Plungė	102
Marijampolė	Klaipėda	286
Vilnius	Klaipėda	286
Miesto pavadinimas	Gyventojų kiekis	
Vilnius	100000	
Klaipėda	1552233	
Marijampolė	15522	
Šiauliai	1545452	
Plungė	155222	
Rezultatai		
Tokių maršrutų nėra.		
Rezultatai (po panaikinimo)		
Tokių maršrutų nėra.		

Ekrano įvestys/rezultatai

Pradiniai duomenys (maršrutai)		
Pirmas miestas	Antras miestas	Atstumas tarp miestų
Vilnius	Klaipėda	286
Marijampolė	Klaipėda	286
Šiauliai	Plungė	102

Pradiniai duomenys (miestai)	
Miesto pavadinimas	Gyventojų kiekis
Vilnius	100000
Klaipėda	1552233
Marijampolė	15522
Šiauliai	1545452
Plungė	155222

Rezultatai
Maršrutų nėra.

3 variantas)

Nėra vieno ar abiejų duomenų failų.

Ekrano kopija

MARŠRUTŲ PAIEŠKOS

Trūksta duomenų failo/failų!

Iveskite pradinį miestą:

Iveskite didžiausią leistiną gyventojų kiekį mieste:

Iveskite minimalų maršruto atstumą kilometrais:

Duomenų failai:

Choose Files No file chosen

Surasti

3.8. Dėstytojo pastabos

Testas: 1/3

Gynimas: 6/6

Ataskaita: 1/1

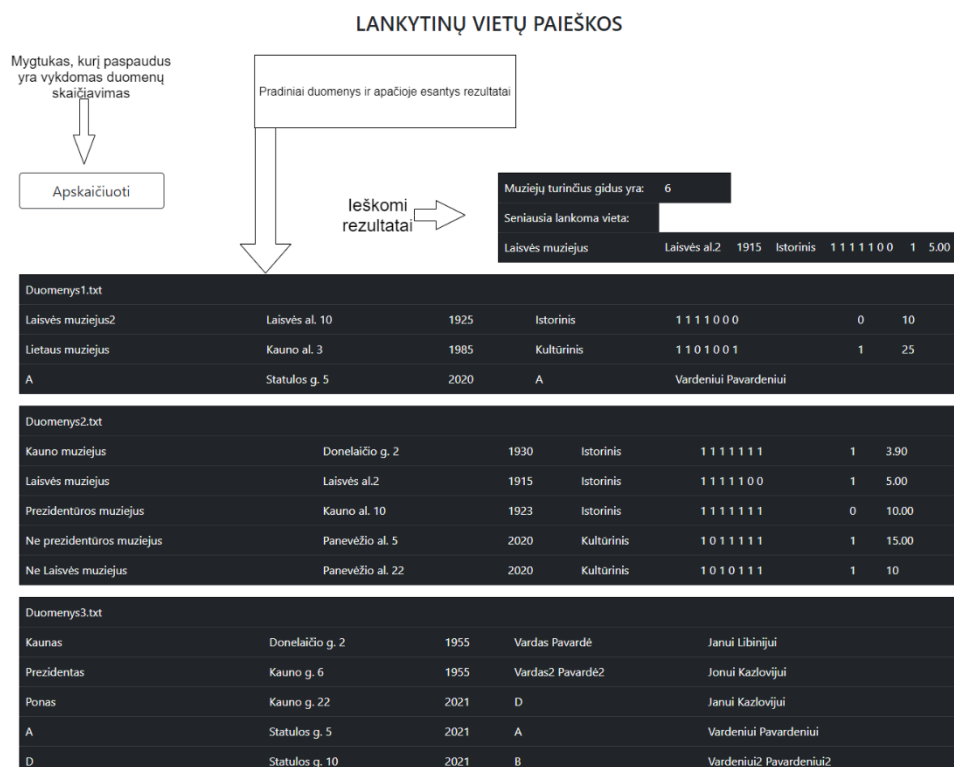
Galutinis įvertinimas: 8.

4. Polimorfizmas ir išimčių valdymas (L4)

4.1. Darbo užduotis

U4_8. Turistų informacijos centras. Turizmo informacijos centre perorganizuoti ir atskirai surašyti duomenys apie kiekviename mieste veikiančius muziejus. Pirmoje eilutėje – miestas, antroje – atsakingo asmens vardas ir pavardė. Turizmo informacijos centras teikia informaciją apie lankytinas vietas – muziejus, paminklus ir kita. Sukurkite abstrakčiąją klasę „Location“ (savybės - pavadinimas, adresas, įkūrimo ar pastatymo metai), kurią paveldės klasės “Museum” (savybės – tipas, 7 savaitės dienos (1 – darbo, 0 – nedarbo), požymis „turi gidą“, bilieto kaina) ir “Statue” (savybės – autorius, kam skirtas). • Suskaičiuokite, kiek muziejų turi gidus, rezultata atspausdinkite ekrane. • Raskite seniausią lankytiną vietą, visą informaciją apie ją atspausdinkite ekrane. • Sudarykite visų lankytinų vietų sąrašą ir įrašykite į failą „VisosVietos.csv“. • Sudarykite ir surikiuokite naujų lankytinų vietų sąrašą, pateikdami pilną informaciją apie jas. Muziejus yra naujas, jei nuo įkūrimo prabėgo mažiau, nei 2 metai. Paminklas yra naujas, jei nuo pastatymo prabėgo mažiau, nei metai. Muziejus rikiuokite pagal bilieto kainas, paminklus – pagal autorius. Rezultatus įrašykite į failą „Nauji.csv“.

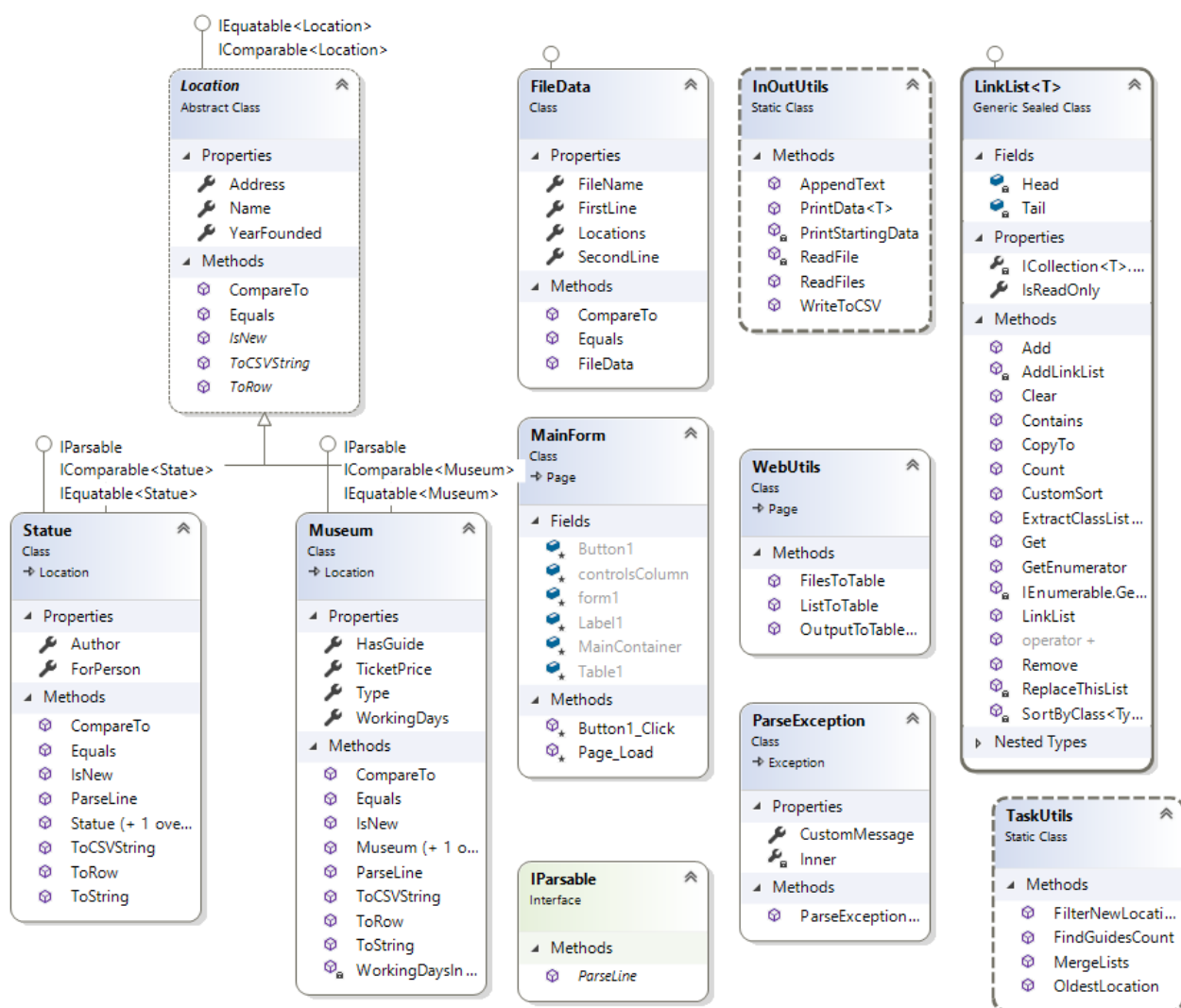
4.2. Grafinės vartotojo sąsajos schema



4.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
#Button1	CssClass	btn btn-outline-dark btn-lg
#Button1	Height	50px
#Button1	Text	Apskaiciuoti
#Button1	Width	200px
#Label1	ForeColor	Red
#Table1	CssClass	table table-hover table-dark
#Table1	GridLines	Both

4.4. Klasių diagrama



4.5. Programos vartotojo vadovas

Programa yra itin paprasta – tereikia paspausti mygtuką „Apskaičiuoti“ ir visi duomenys bus apskaičiuoti bei išvesti į rezultatų dokumentą („Rezultatai.txt“).

Norint sukurti duomenų dokumentus, reikia:

- Pirmoje eilutėje parašyti turizmo informacijos centro miestą.
- Antroje – Atsakingo asmens vardą ir pavardę.
- Nuo trečios iki kiek naudotojas nori, eina lankomų vietų duomenys (iš anksto nustatyta skyryba yra „;“):
 - Muziejų duomenų sudarymas:
 - Pavadinimas;Adresas;Įkūrimo metai;Tipas;7 savaitės dienos (0 – nedarbo diena, 1 – darbo) (skiriama tarpais iki kabliataškio);(0 – neturi gido, 1 – turi gidą);Bilieto kaina
 - Statulų duomenų sudarymas:
 - Pavadinimas;Adresas;Įkūrimo metai;Autorius;Kam skirtas
- Pradinių duomenų dokumentai (kiekis neribojamas) yra dedami į „App_Data“ aplanką.
- Rezultatai randami ekrane ir tekstiniame dokumente „App_Data“ aplanke, pavadinimu „Rezultatai.txt“.
- Kiti rezultatai yra randami, taip pat „App_Data“ aplanke, pavadinimais „Nauji.csv“ ir „VisosVietos.csv“.

4.6. Programos tekstas

IParsable.cs

```
using System;
namespace Lab4_WebApp
{
    public interface IParsable
    {
        //interface for parsing data to class objects
        void ParseLine(string[] Parts);
    }
}
```

Location.cs

```
using System;
using System.Web.UI.WebControls;
namespace Lab4_WebApp
{
    public abstract class Location : IEquatable<Location>, IComparable<Location>
    {
        //Abstract class
        public string Name { get; set; }
        public string Address { get; set; }
        public int YearFounded { get; set; }

        public int CompareTo(Location other)
        {
            throw new NotImplementedException();
        }
    }
}
```

```

        public bool Equals(Location other)
        {
            return Name == other.Name && Address == other.Address && YearFounded ==
other.YearFounded;
        }

        /// <summary>
        /// Abstract method for printing to CSV file
        /// </summary>
        /// <returns>string line</returns>
        public abstract string ToCSVString();

        /// <summary>
        /// Abstract method for output to table
        /// </summary>
        /// <returns>TableRow object</returns>
        public abstract TableRow ToRow();

        /// <summary>
        /// Abstract method, checks if location is new or not (by a custom criteria)
        /// </summary>
        /// <returns>true or false, depending on the outcome</returns>
        public abstract bool IsNew();
    }
}

```

Museum.cs

```

using System;
using System.Linq;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Lab4_WebApp
{
    public class Museum : Location, IParsable, IComparable<Museum>, IEquatable<Museum>
    {
        public string Type { get; set; }
        private int[] WorkingDays { get; set; }
        public bool HasGuide { get; set; }
        public decimal TicketPrice { get; set; }

        //Public constructor
        public Museum(string name, string address, int year, string type, int[] workingDays,
bool guide, decimal ticketPrice)
        {
            Name = name;
            Address = address;
            YearFounded = year;
            Type = type;
            WorkingDays = workingDays;
            HasGuide = guide;
            TicketPrice = ticketPrice;
        }

        //Empty constructor
        public Museum() { }

        //Parses line into this class object's properties
        public void ParseLine(string[] Parts)
        {
            try
            {
                Name = Parts[0];
                Address = Parts[1];
                YearFounded = int.Parse(Parts[2]);
            }
            catch { }
        }
    }
}

```

```

        Type = Parts[3];
        string[] split = Parts[4].Split(' ');
        WorkingDays = split.Where(s => int.TryParse(s, out int _)).Select(s =>
int.Parse(s)).ToArray();
        HasGuide = int.Parse(Parts[5]) == 1 ? true : false;
        TicketPrice = decimal.Parse(Parts[6]);
    }

    catch (Exception ex)
    {
        throw new ParseException("Klaidingas nuskaitymas!", ex);
    }
}

/// <summary>
/// For convenience, makes int array into a string line
/// </summary>
/// <returns></returns>
private string WorkingDaysIntoString()
{
    string line = "";
    for (int i = 0; i < WorkingDays.Length; i++)
    {
        if (i == WorkingDays.Length - 1) line += (WorkingDays[i].ToString());
        else
        {
            line += (WorkingDays[i].ToString() + " ");
        }
    }
    return line;
}

//IComparable<Museum> realisation
public int CompareTo(Museum other)
{
    return TicketPrice.CompareTo(other.TicketPrice);
}

//Outputs data into line string
public override string ToString()
{
    string line = WorkingDaysIntoString();

    return String.Format("|{0, -25}|{1, -20}|{2, 20}|{3, -10}|{4, 20}|{5, 10}|{6,
15}|", Name, Address, YearFounded, Type, line, (HasGuide ? "1" : "0"), TicketPrice);
}

//Outputs data into string made for CSV files
public override string ToCSVString()
{
    string line = WorkingDaysIntoString();

    return String.Format("{0, -25};{1, -20};{2, 20};{3, -10};{4, 20};{5, 10};{6, 15}",
Name, Address, YearFounded, Type, line, (HasGuide ? "1" : "0"), TicketPrice);
}

//Outputs data into TableRow objects
public override TableRow ToRow()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell() { Text = Name });
    row.Cells.Add(new TableCell() { Text = Address });
    row.Cells.Add(new TableCell() { Text = YearFounded.ToString() });
    row.Cells.Add(new TableCell() { Text = Type });
    row.Cells.Add(new TableCell() { Text = WorkingDaysIntoString() });
    row.Cells.Add(new TableCell() { Text = (HasGuide ? "1" : "0") });
    row.Cells.Add(new TableCell() { Text = TicketPrice.ToString() });
    return row;
}

```

```

    }
    /// <summary>
    /// Equals method
    /// </summary>
    /// <param name="other">other class object to compare</param>
    /// <returns></returns>
    public bool Equals(Museum other)
    {
        return Name == other.Name && Address == other.Address && YearFounded ==
other.YearFounded && TicketPrice == other.TicketPrice && Type == other.Type;
    }

    /// <summary>
    /// Checks if the location is new or not
    /// </summary>
    /// <returns>true or false</returns>
    public override bool IsNew()
    {
        int currentYear = DateTime.Now.Year;
        return (currentYear - YearFounded < 2 ? true : false);
    }
}
}

```

Statue.cs

```

using System;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace Lab4_WebApp
{
    public class Statue : Location, IParsable, IComparable<Statue>, IEquatable<Statue>
    {
        public string Author { get; set; }
        public string ForPerson { get; set; }
        //Constructor
        public Statue(string name, string address, int year, string author, string person)
        {
            Name = name;
            Address = address;
            YearFounded = year;
            Author = author;
            ForPerson = person;
        }

        public Statue() { }

        //Parses line into this class object's properties
        public void ParseLine(string[] Parts)
        {
            try
            {
                Name = Parts[0];
                Address = Parts[1];
                YearFounded = int.Parse(Parts[2]);
                Author = Parts[3];
                ForPerson = Parts[4];
            }
            catch (Exception ex)
            {
                throw new ParseException("Klaidingas nuskaitymas!", ex);
            }
        }

        /// <summary>

```



```

/// Returns TableRow object with this object's properties
/// </summary>
/// <returns>TableRow object</returns>
public override TableRow ToRow()
{
    TableRow row = new TableRow();
    row.Cells.Add(new TableCell() { Text = Name });
    row.Cells.Add(new TableCell() { Text = Address });
    row.Cells.Add(new TableCell() { Text = YearFounded.ToString() });
    row.Cells.Add(new TableCell() { Text = Author });
    row.Cells.Add(new TableCell() { Text = ForPerson, ColumnSpan = 3 });
    return row;
}
//Comparable interface method realisation
public int CompareTo(Statue other)
{
    return (Author.CompareTo(other.Author) * -1);
}
//ToString method override
public override string ToString()
{
    return String.Format("|{0, -25}|{1, -20}|{2, 20}|{3, -25}|{4, -25}|", Name,
Address, YearFounded, Author, ForPerson);
}
/// <summary>
/// Outputs class object into string line for CSV file
/// </summary>
/// <returns>a string line</returns>
public override string ToCSVString()
{
    return String.Format("{0, -25};{1, -20};{2, 20};{3, -25};{4, -25}", Name, Address,
YearFounded, Author, ForPerson);
}

/// <summary>
/// Checks if two class objects are the same
/// </summary>
/// <param name="other">other object to check</param>
/// <returns>a true or false statement</returns>
public bool Equals(Statue other)
{
    return Name == other.Name && Address == other.Address && YearFounded ==
other.YearFounded && Author == other.Author && ForPerson == other.ForPerson;
}

/// <summary>
/// Checks if a location is new or old
/// </summary>
/// <returns>true or false, depending on if location is new or not</returns>
public override bool IsNew()
{
    int currentYear = DateTime.Now.Year;
    return (currentYear - YearFounded < 1 ? true : false);
}
}
}

```

FileData.cs

```

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

namespace Lab4_WebApp

```

```

{
    public class FileData : IEquatable<FileData>, IComparable<FileData>
    {
        //Properties
        public string FirstLine { get; set; }
        public string SecondLine { get; set; }
        private LinkedList<Location> Locations { get; set; }
        public string FileName { get; set; }

        //Constructor
        public FileData(string firstLine, string secondLine, LinkedList<Location> locations,
string fileName)
        {
            FirstLine = firstLine;
            SecondLine = secondLine;
            Locations = locations;
            FileName = fileName;
        }

        public LinkedList<Location> GetLocationsList()
        {
            return Locations;
        }

        //Equals interface method
        public bool Equals(FileData other)
        {
            throw new NotImplementedException();
        }

        //CompareTo interface method
        public int CompareTo(FileData other)
        {
            throw new NotImplementedException();
        }
    }
}

```

LinkedList.cs

```

using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab4_WebApp
{
    public sealed class LinkedList<T> : ICollection<T> where T : IEquatable<T>, IComparable<T>
    {
        private sealed class Node<Type> //Private generic Node class
        {
            public Type Value { get; set; }
            public Node<Type> Link { get; set; }

            public Node(Type value, Node<Type> link)
            {
                Value = value;
                Link = link;
            }

            public Node() { }
        }

        private Node<T> Head;
        private Node<T> Tail;
    }
}

```

```

//Constructor
public LinkedList()
{
    Head = null;
    Tail = null;
}

/// <summary>
/// Outputs how many objects are in the list
/// </summary>
/// <returns>amount of objects</returns>
public int Count()
{
    int count = 0;
    foreach(T type in this)
    {
        count++;
    }
    return count;
}

/// <summary>
/// Tells if this list is read only
/// </summary>
public bool IsReadOnly => false;

/// <summary>
/// ICollection interface .Count property
/// </summary>
int ICollection<T>.Count => Count();

/// <summary>
/// Adds a new object to the list
/// </summary>
/// <param name="item"></param>
public void Add(T item)
{
    Node<T> node = new Node<T>(item, null);

    if (Head == null)
    {
        Head = node;
        Tail = node;
    }

    else
    {
        Tail.Link = node;
        Tail = node;
    }
}

/// <summary>
/// Gets an object by index
/// </summary>
/// <param name="index">object index to return</param>
/// <returns>object by index in the list</returns>
public T Get(int index)
{
    int count = 0;
    for (Node<T> d = Head; d != null; d = d.Link)
    {
        if (count == index)
        {

```

```

        return d.Value;
    }
    count++;
}
return default(T);
}

/// <summary>
/// Resets list
/// </summary>
public void Clear()
{
    Head = null;
    Tail = null;
}

/// <summary>
/// Checks if a list contains a specific object
/// </summary>
/// <param name="item">object to check for</param>
/// <returns>true or false, depending on the outcome</returns>
public bool Contains(T item)
{
    for (Node <T> d = Head; d != null; d = d.Link)
    {
        if (d.Value.Equals(item)) return true;
    }
    return false;
}

/// <summary>
/// Copies this list to the same type array
/// </summary>
/// <param name="array">array to copy to</param>
/// <param name="arrayIndex">array's index (free space to assign data)</param>
public void CopyTo(T[] array, int arrayIndex)
{
    try
    {
        for (Node<T> d = Head; d != null; d = d.Link)
        {
            array[arrayIndex++] = d.Value;
        }
    }

    catch (Exception)
    {
        throw;
    }
}

/// <summary>
/// Adds LinkedList to this LinkedList
/// </summary>
/// <param name="toAdd">list to add</param>
private void AddLinkedList(LinkedList<T> toAdd)
{
    foreach(T value in toAdd)
    {
        Add(value);
    }
}

/// <summary>
/// Replaces this list with another list
/// </summary>

```

```

/// <param name="toReplace">list to replace this list</param>
private void ReplaceThisList(LinkList<T> toReplace)
{
    Clear();
    foreach (T value in toReplace)
    {
        Add(value);
    }
}

/// <summary>
/// Removes a particular object
/// </summary>
/// <param name="item">object to remove</param>
/// <returns>true if remove succeeded, else - false</returns>
public bool Remove(T item)
{
    for (Node<T> d = Head; d != null; d = d.Link)
    {
        if (d.Value.Equals(item) && d == Head)
        {
            Head = d.Link;
            return true;
        }

        else if (d.Value.Equals(item) && d != Head)
        {
            Node<T> chargeNode;
            for (chargeNode = Head; chargeNode.Link != d; chargeNode =
chargeNode.Link) ;
            chargeNode.Link = d.Link;
            return true;
        }
    }
    return false;
}

/// <summary>
/// Extracts a particular inherited class from parent's list
/// </summary>
/// <typeparam name="Type">type of the list to extract</typeparam>
/// <returns>extracted list by class</returns>
public LinkList<T> ExtractClassList<Type>()
{
    LinkList<T> extracted = new LinkList<T>();
    for (Node<T> d = Head; d != null; d = d.Link)
    {
        if (d.Value is Type)
        {
            extracted.Add(d.Value);
        }
    }
    return extracted;
}

/// <summary>
/// Sorts particular class objects by a defined pattern
/// </summary>
/// <typeparam name="Type">Class to sort like</typeparam>
private void SortByClass<Type>() where Type : class, IComparable<Type>
{
    bool flag = true;
    while (flag)
    {
        flag = false;
        for (Node<T> d = Head; d.Link != null; d = d.Link)

```

```

        {
            Type value = d.Value as Type;
            Type valueLink = d.Link.Value as Type;
            if (value.CompareTo(valueLink) < 0)
            {
                T holder = d.Value;
                d.Value = d.Link.Value;
                d.Link.Value = holder;
            }
        }
    }
}

/// <summary>
/// Executes extraction of lists and their sorting
/// </summary>
public void CustomSort()
{
    LinkedList<T> museums = ExtractClassList<Museum>();
    LinkedList<T> statues = ExtractClassList<Statue>();

    if (museums.Count() > 0) museums.SortByClass<Museum>();
    if (statues.Count() > 0) statues.SortByClass<Statue>();

    ReplaceThisList(museums + statues);
}

/// <summary>
/// Operator "+" method
/// </summary>
/// <param name="one">List to add</param>
/// <param name="second">List to add</param>
/// <returns>connected list</returns>
public static LinkedList<T> operator +(LinkedList<T> one, LinkedList<T> second)
{
    LinkedList<T> merged = new LinkedList<T>();
    merged.AddLinkedList(one);
    merged.AddLinkedList(second);
    return merged;
}

public IEnumerator<T> GetEnumerator()
{
    for (Node<T> d = Head; d != null; d = d.Link)
    {
        yield return d.Value;
    }
}

IEnumerator IEnumerable.GetEnumerator()
{
    throw new NotImplementedException();
}
}
}

```

InOutUtils.cs

```

using System;

using System.IO;

using System.Text;

```

```

namespace Lab4_WebApp
{
    public static class InOutUtils
    {
        /// <summary>
        /// Reads all files in the "App_Data" folder
        /// </summary>
        /// <param name="fileNames"></param>
        /// <returns></returns>
        public static LinkedList<FileData> ReadFiles(string[] fileNames)
        {
            LinkedList<FileData> files = new LinkedList<FileData>();

            try
            {
                foreach (string file in fileNames)
                {
                    files.Add(ReadFile(file));
                }
            }
            catch (Exception)
            {
                throw new ParseException();
            }

            return files;
        }

        /// <summary>
        /// Reads one file individually
        /// </summary>
        /// <param name="fileName">name of the file</param>
        /// <returns>a FileData object to hold all the data from the file</returns>
        private static FileData ReadFile(string fileName)
        {

```

```

LinkedList<Location> LocationsList = new LinkedList<Location>();

string [] information = new string[2];

using (StreamReader input = new StreamReader(fileName, Encoding.UTF8))
{
    try
    {
        information[0] = input.ReadLine();
        information[1] = input.ReadLine();
        string line;
        while ((line = input.ReadLine()) != null)
        {
            string[] Parts = line.Split(';');
            int amountOfParts = Parts.Length;
            switch (amountOfParts)
            {
                case 7:
                    var obj1 = new Museum();
                    obj1.ParseLine(Parts);
                    LocationsList.Add(obj1);
                    break;
                case 5:
                    var obj2 = new Statue();
                    obj2.ParseLine(Parts);
                    LocationsList.Add(obj2);
                    break;
            }
        }
    }
    catch (Exception)
    {
        throw new ParseException();
    }
}

```



```

        FileData newFile = new FileData(information[0], information[1],
LocationsList, fileName);

        return newFile;
    }

    /// <summary>
    /// Outputs list into CSV file
    /// </summary>
    /// <param name="fileName">name of the file to write to</param>
    /// <param name="list">list to output</param>
    public static void WriteToCSV(string fileName, LinkList<Location> list)
    {
        using (StreamWriter output = new StreamWriter(fileName, true,
Encoding.UTF8))
        {
            try
            {
                if (list.Count() == 0) throw new NullReferenceException();

                foreach (Location location in list)
                {
                    output.WriteLine(location.ToCSVString());
                }
            }
            catch (Exception)
            {
                output.WriteLine("Sąrašas yra tuščias.");
            }
        }
    }

    /// <summary>
    /// Prints data into results file
    /// </summary>
    /// <param name="fileName">Filename to print into</param>

```

```

    /// <param name="files">List of files to print from</param>

    /// /// <param name="header">name to call this list</param>

    public static void PrintData(string fileName, LinkList<FileData> files,
    string header)
    {
        using (StreamWriter output = new StreamWriter(fileName, true,
    Encoding.UTF8))
        {
            try
            {
                output.WriteLine(header);

                foreach (FileData file in files)
                {
                    string[] fileNameParts = file.FileName.Split('\\');

                    output.WriteLine("Dokumento pavadinimas: " +
    fileNameParts[fileNameParts.Length - 1]);

                    output.WriteLine(file.FirstLine);

                    output.WriteLine(file.SecondLine);

                    LinkList<Location> museums =
    file.GetLocationsList().ExtractClassList<Museum>();

                    LinkList<Location> statues =
    file.GetLocationsList().ExtractClassList<Statue>();

                    LinkList<Location> pointer = museums;

                    for (int i = 0; i < 2; i++)
                    {
                        if (i == 0 && pointer.Count() > 0)
                        {
                            string line = String.Format("|{0, -25}|{1, -
    20}|{2, 20}|{3, -10}|{4, 20}|{5, -10}|{6, -15}|", "Pavadinimas", "Adresas",
    "Įkūrimo metai", "Tipas", "Darbo dienos", "Turi gida?", "Bilieto kaina");

                            output.WriteLine(new string('-', line.Length));

                            output.WriteLine(line);

                            output.WriteLine(new string('-', line.Length));

                        }

                        else if (i == 1 && pointer.Count() > 0)

```

```

        {
            string line = String.Format("|{0, -25}|{1, -20}|{2, 20}|{3, -25}|{4, -25}|", "Pavadinimas", "Adresas", "Įkūrimo metai", "Autorius", "Skirta");

            output.WriteLine();

            output.WriteLine(new string('-', line.Length));

            output.WriteLine(line);

            output.WriteLine(new string('-', line.Length));
        }

        foreach (Location location in pointer)
        {
            if (i == 0)
            {
                output.WriteLine(location.ToString());
            }

            else
            {
                output.WriteLine(location.ToString());
            }
        }

        pointer = statues;
    }

    output.WriteLine();
}

catch (NullReferenceException)
{
    output.WriteLine("Sarašas yra tuščias.");
}

}

}

/// <summary>

```

```

    /// Prints data into results file

    /// </summary>

    /// <param name="fileName">Filename to print into</param>

    /// <param name="files">List of locations to print from</param>

    /// <param name="header">name to call this list</param>

    public static void PrintData(string fileName, LinkList<Location>
locations, string header)

    {

        using (StreamWriter output = new StreamWriter(fileName, true,
Encoding.UTF8))

        {

            try

            {

                if (locations.Count() == 0) throw new ParseException();

                output.WriteLine(header);

                LinkList<Location> museums =
locations.ExtractClassList<Museum>();

                LinkList<Location> statues =
locations.ExtractClassList<Statue>();

                LinkList<Location> pointer = museums;

                for (int i = 0; i < 2; i++)

                {

                    if (pointer.Count() == 0 && i == 0) pointer = statues;

                    if (i == 0)

                    {

                        string line = String.Format("|{0, -25}|{1, -20}|{2,
20}|{3, -10}|{4, 20}|{5, -10}|{6, -15}|", "Pavadinimas", "Adresas", "Įkūrimo
metai", "Tipas", "Darbo dienos", "Turi gida?", "Bilieto kaina");

                        output.WriteLine(new string('-', line.Length));

                        output.WriteLine(line);

                        output.WriteLine(new string('-', line.Length));

                    }

                    else if (i == 1 && pointer.Count() != 0)

                    {

```

```

        string line = String.Format("|{0, -25}|{1, -20}|{2, 20}|{3, -25}|{4, -25}|", "Pavadinimas", "Adresas", "Įkūrimo metai", "Autorius", "Skirta");

        output.WriteLine();

        output.WriteLine(new string('-', line.Length));

        output.WriteLine(line);

        output.WriteLine(new string('-', line.Length));
    }

    foreach (Location location in pointer)
    {
        if (i == 0)
        {
            output.WriteLine(location.ToString());
        }

        else
        {
            output.WriteLine(location.ToString());
        }
    }

    pointer = statues;
}

output.WriteLine();
}

catch (Exception)
{
    output.WriteLine("Sąrašas yra tuščias.");
}

}

}

/// <summary>
/// Appends text to results file

```

```

        /// </summary>

        /// <param name="fileName">name of the file to output to</param>

        /// <param name="lines">lines to output</param>

        public static void AppendText(string fileName, string[] lines)
        {
            using (StreamWriter output = new StreamWriter(fileName, true,
Encoding.UTF8))
            {
                for (int i = 0; i < lines.Length; i++)
                {
                    output.WriteLine(lines[i]);
                }
            }
        }
    }
}

```

TaskUtils.cs

```

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

namespace Lab4_WebApp
{
    public static class TaskUtils
    {
        /// <summary>
        /// Finds guides count throughout all lists
        /// </summary>
        /// <param name="filesList"></param>
        /// <returns></returns>
        public static int FindGuidesCount(LinkedList<FileData> filesList)
        {
            int count = 0;
            try
            {
                foreach (FileData file in filesList)
                {
                    foreach (Location location in file.GetLocationsList())
                    {
                        if (location is Museum && ((Museum)location).HasGuide) count++;
                    }
                }
            }
            catch (Exception)
            {
                throw new NullReferenceException();
            }
            return count;
        }
    }
}

```

```

}

/// <summary>
/// Finds the oldest location in all of the lists
/// </summary>
/// <param name="files">All files list</param>
/// <returns>oldest location</returns>
public static Location OldestLocation(LinkList<FileData> files)
{
    Location oldest = new Statue("", "", DateTime.Now.Year, "", "");
    try
    {
        foreach (FileData file in files)
        {
            foreach (Location location in file.GetLocationsList())
            {
                if (oldest.YearFounded > location.YearFounded)
                {
                    oldest = location;
                }
            }
        }
        if (oldest == new Statue("", "", DateTime.Now.Year, "", "")) oldest = null;
    }
    catch (Exception)
    {
        throw new NullReferenceException();
    }
    return oldest;
}

/// <summary>
/// Filters new locations to a new list
/// </summary>
/// <param name="files">list of all files</param>
/// <returns>LinkList of Location type, made out of new locations</returns>
public static LinkList<Location> FilterNewLocations(LinkList<FileData> files)
{
    LinkList<Location> filtered = new LinkList<Location>();
    int currentYear = DateTime.Now.Year;
    try
    {
        foreach (FileData file in files)
        {
            foreach (Location location in file.GetLocationsList())
            {
                if (location.IsNew())
                {
                    filtered.Add(location);
                }
            }
        }
    }
    catch (Exception)
    {
        throw new NullReferenceException();
    }

    return filtered;
}

/// <summary>
/// Merges all lists out of all the files into one Location list
/// </summary>
/// <param name="files">all files list</param>
/// <returns>Location class link list</returns>

```

```

public static LinkList<Location> MergeLists(LinkList<FileData> files)
{
    LinkList<Location> all = new LinkList<Location>();
    try
    {
        foreach (FileData file in files)
        {
            foreach (Location location in file.GetLocationsList())
            {
                all.Add(location);
            }
        }
    }
    catch (Exception)
    {
        throw new NullReferenceException();
    }

    return all;
}
}
}

```

WebUtils.cs

```

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

namespace Lab4_WebApp
{
    public class WebUtils : System.Web.UI.Page
    {
        /// <summary>
        /// Outputs data into a table
        /// </summary>
        /// <param name="table">Table to output data</param>
        /// <param name="columnSpan">span of the header row's cell's column</param>
        /// <param name="output">data to output</param>
        /// <param name="outputLine">header line</param>
        public static void OutputToTableRow(Table table, int columnSpan, int output, string
outputLine)
        {
            TableRow row = new TableRow();
            row.Cells.Add(new TableCell() { Text = outputLine, ColumnSpan = columnSpan });
            row.Cells.Add(new TableCell() { Text = output.ToString() });
            table.Rows.Add(row);
        }

        /// <summary>
        /// Makes list into a table
        /// </summary>
        /// <param name="controls">Controls collection to put table in</param>
        /// <param name="locations">list of locations to output to a table</param>
        /// <param name="headerRow">header row</param>
        public static void ListToTable(ControlCollection controls, LinkList<Location>
locations, TableRow headerRow)
        {
            Table table = new Table()
            {
                GridLines = GridLines.Both,
                CssClass = "table table-hover table-dark"
            };

```



```

        table.Rows.Add(headerRow);
    try
    {
        if (locations.Count() == 0) throw new ParseException();
        foreach (Location location in locations)
        {
            table.Rows.Add(location.ToRow());
        }
    }
    catch (ParseException)
    {
        TableRow error = new TableRow();
        table.Rows.Add(error);
        table.Rows[table.Rows.GetRowIndex(error)].Cells.Add(new TableCell() { Text =
"Tuščias sąrašas.", ColumnSpan = 7 });
    }
    finally
    {
        controls.Add(table);
    }
}

/// <summary>
/// Executes files into Location lists and later - into tables
/// </summary>
/// <param name="controls">place to add table to</param>
/// <param name="files">list of all the files</param>
public static void FilesToTable(ControlCollection controls, LinkList<FileData> files)
{
    foreach (FileData file in files)
    {
        TableRow fileNameHeader = new TableRow();
        string[] fileNameParts = file.FileName.Split('\\');
        fileNameHeader.Cells.Add(new TableCell() { Text =
fileNameParts[fileNameParts.Length - 1], ColumnSpan = 7});

        ListToTable(controls, file.GetLocationsList(), fileNameHeader);
    }
}
}
}

```

ParseException.cs

```

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

namespace Lab4_WebApp
{
    public class ParseException : Exception
    {
        public string CustomMessage { get; set; }
        Exception Inner { get; set; }
        public ParseException() { }
        //Constructors
        public ParseException(string customMessage)
        {
            CustomMessage = customMessage;
        }
    }
}

```

```

        public ParseException(string customMessage, Exception inner) : base(customMessage,
inner)
        {
            CustomMessage = customMessage;
            Inner = inner;
        }
    }
}

MainForm.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 Lab4_WebApp
{
    public partial class MainForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = "";
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            const string CFrAll = "App_Data\\VisosVietos.csv";
            const string CFrNew = "App_Data\\Nauji.csv";
            const string CFr = "App_Data\\Rezultatai.txt"; //file names

            try
            {
                if (File.Exists(Server.MapPath(CFrAll))) File.Delete(Server.MapPath(CFrAll));
//deletes files, to prevent duplication
                if (File.Exists(Server.MapPath(CFrNew))) File.Delete(Server.MapPath(CFrNew));
                if (File.Exists(Server.MapPath(CFr))) File.Delete(Server.MapPath(CFr));

                string[] Files = Directory.GetFiles(Server.MapPath("App_Data/")).Where(s =>
s.EndsWith(".txt")) //gets all the fileNames from App_Data folder
                    .ToArray();
                if (Files.Count() == 0) throw new ParseException("Nėra duomenų failų!");
                LinkedList<FileData> allLists = InOutUtils.ReadFiles(Files);

                LinkedList<Location> AllDataList = TaskUtils.MergeLists(allLists); //merges all
files into one list

                InOutUtils.WriteToCSV(Server.MapPath(CFrAll), AllDataList);
                LinkedList<Location> newLocations = TaskUtils.FilterNewLocations(allLists);
                newLocations.CustomSort();
                InOutUtils.WriteToCSV(Server.MapPath(CFrNew), newLocations);
                InOutUtils.PrintData(Server.MapPath(CFr), allLists, "Pradiniai duomenys");
                InOutUtils.PrintData(Server.MapPath(CFr), AllDataList, "Visų lankytinų vietų
sąrašas.");
                InOutUtils.PrintData(Server.MapPath(CFr), newLocations, "Naujų vietų
sąrašas.");

                int guidesCount = TaskUtils.FindGuidesCount(allLists);
                Location oldestLocation = TaskUtils.OldestLocation(allLists);

                WebUtils.OutputToTableRow(Table1, 1, guidesCount, "Muziejų turinčius gidus
yra:");
                Table1.Rows.Add(new TableRow());
            }
            catch { }
        }
    }
}

```

```

        Table1.Rows[1].Cells.Add(new TableCell() { Text = "Seniausia lankoma vieta:"
    });
    Table1.Rows.Add(oldestLocation.ToRow());
    Table1.Visible = true; //Outputs results

    string[] AppendLines = new string[2];
    AppendLines[0] = String.Format("Muziejų turinčius gidus yra: {0}",
guidesCount);
    AppendLines[1] = String.Format("Seniausia lankoma vieta: {0}", (oldestLocation
== null ? "Sąrašas tuščias." : oldestLocation.ToString()));
    InOutUtils.AppendText(Server.MapPath(CFr), AppendLines); //Adds additional
data to the results file

    WebUtils.FilesToTable(MainContainer.Controls, allLists);
    TableRow headerRowAll = new TableRow();
    headerRowAll.Cells.Add(new TableCell() { Text = "Pilnas sąrašas.", ColumnSpan
= 7 }); //Outputs results
    WebUtils.ListToTable(MainContainer.Controls, AllDataList, headerRowAll);

    TableRow headerRowNew = new TableRow();
    headerRowNew.Cells.Add(new TableCell() { Text = "Naujų lankytinų vietų
sąrašas.", ColumnSpan = 7 });
    WebUtils.ListToTable(MainContainer.Controls, newLocations, headerRowNew);
//Outputs results
    }

    catch (ParseException ex)
    {
        if (ex.CustomMessage != null)
        {
            Label1.Text = ex.CustomMessage; //creates custom error
        }
        else
        {
            Label1.Text = "Nuskaitymo klaida!"; //creates custom error
        }
        Table1.Visible = false;
    }

    catch (NullReferenceException)
    {
        Label1.Text = "Klaida! Skaičiuojama su nuliu!"; //creates custom error
    }

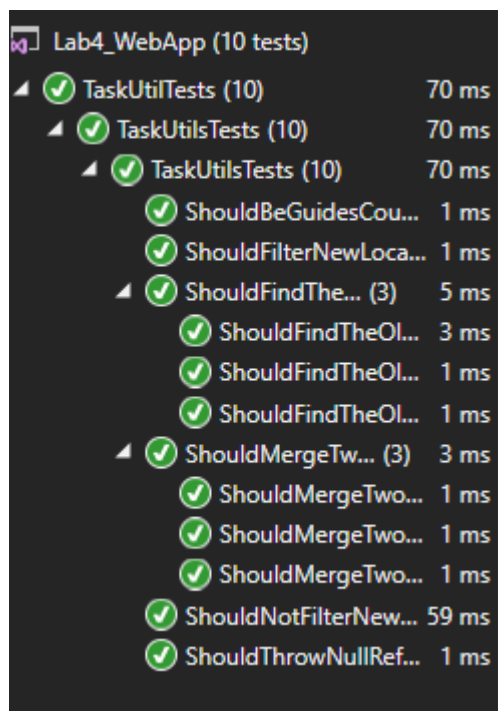
    catch (Exception)
    {
        Label1.Text = "Nenustatyta klaida!"; //creates custom error
    }

    }

}
}

```

4.7. Pradiniai duomenys ir rezultatai



1 pav. (Įvykdyti testai)

Pirmas pradinių duomenų variantas

Duomenys1.txt

Kaunas

Ponas Vardenis

Laisvės muziejus2;Laisvės al. 10;1925;Istorinis;1 1 1 0 0 0;0;10

Lietaus muziejus;Kauno al. 3;1985;Kultūrinis;1 1 0 1 0 0 1;1;25

A;Statulos g. 5;2020;A;Vardeniui Pavardeniui

Duomenys2.txt

Kaunas

Džėjus Getsbis

Kauno muziejus;Donelaičio g. 2;1930;Istorinis;1 1 1 1 1 1 1;1;3.90

Laisvės muziejus;Laisvės al.2;1915;Istorinis;1 1 1 1 1 0 0;1;5.00

Prezidentūros muziejus;Kauno al. 10;1923;Istorinis;1 1 1 1 1 1 1;0;10.00

Ne prezidentūros muziejus;Panevėžio al. 5;2020;Kultūrinis;1 0 1 1 1 1 1;1;15.00

Ne Laisvės muziejus;Panevėžio al. 22;2020;Kultūrinis;1 0 1 0 1 1 1;1;10

Duomenys3.txt

Kaunas

Džėjus Getsbis

Kaunas;Donelaičio g. 2;1955;Vardas Pavardė;Janui Libinijui

Prezidentas;Kauno g. 6;1955;Vardas2 Pavardė2;Jonui Kazlovijui

Ponas;Kauno g. 22;2021;D;Janui Kazlovijui

A;Statulos g. 5;2021;A;Vardeniui Pavardeniui

D;Statulos g. 10;2021;B;Vardeniui2 Pavardeniui2

Super Laisvės muziejus;Panevėžio al. 22;2020;Kultūrinis;1 0 1 0 1 1 1;1;8

Rezultatai

Apskaiciuoti

Muziejų turinčius gідus yra: 6

Seniausia lankoma vieta:

Laisvės muziejus Laisvės al.2 1915 Istorinis 1 1 1 1 0 0 1 5.00

Duomenys1.txt						
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25
A	Statulos g. 5	2020	A	Vardeniui Pavardeniui		

Duomenys2.txt						
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1	1	10

Duomenys3.txt						
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui		
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui		
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui		
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2		
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8

2 pav. (Rezultatai ekrane)

Pilnas sąrašas.						
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25
A	Statulos g. 5	2020	A	Vardeniui Pavardeniui		
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1	1	10
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui		
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui		
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui		
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2		
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8

Naujų lankytinų vietų sąrašas.						
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1	1	10
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2		
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui		

3 pav. (Rezultatai ekrane)

Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1	1	10
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1	1	8
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2		
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui		

4 pav. (Nauji.csv rezultatai)

Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1	1	25
A	Statulos g. 5	2020	A	Vardeniui Pavardeniui		
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1	1	5.00
Prezidentūros muz	Kauno al. 10	1923	Istorinis	1 1 1 1	0	10.00
Ne prezidentūros r	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1	1	15.00
Ne Laisvės muzieju	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0	1	10
Kaunas	Donelaičio g. 2	1955	Vardas Pa	Janui Libinijui		
Prezidentas	Kauno g. 6	1955	Vardas2 P	Jonui Kazlovijui		
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui		
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2		
Super Laisvės muzi	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0	1	8

5 pav. (Nauji.csv rezultatai)

Pradiniai duomenys
Dokumento pavadinimas: Duomenys1.txt
Kaunas
Ponas Vardenis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
A	Statulos g. 5	2020	A	Vardeniui Pavardeniui

Dokumento pavadinimas: Duomenys2.txt
Kaunas
Džėjus Getsbis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	10

Dokumento pavadinimas: Duomenys3.txt
Kaunas
Džėjus Getsbis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2

Visų lankytinų vietų sąrašas.

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	10
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
-------------	---------	---------------	----------	--------

6 pav. (Rezultatai.txt dokumentas)

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
A	Statulos g. 5	2020	A	Vardeniui Pavardeniui
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2

Naujų vietų sąrašas.

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Ne prezidentūros muziejus	Panevėžio al. 5	2020	Kultūrinis	1 0 1 1 1 1 1	1	15.00
Ne laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	10
Super Laisvės muziejus	Panevėžio al. 22	2020	Kultūrinis	1 0 1 0 1 1 1	1	8

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
A	Statulos g. 5	2021	A	Vardeniui Pavardeniui
D	Statulos g. 10	2021	B	Vardeniui2 Pavardeniui2
Ponas	Kauno g. 22	2021	D	Janui Kazlovijui

Muziejų turinčius gidus yra: 6

Seniausia lankoma vieta: Laisvės muziejus | Laisvės al.2 | 1915 | Istorinis | 1 1 1 1 1 0 0 | 1 | 5.00

7 pav. (Rezultatai.txt dokumentas)

Antras duomenų variantas (nėra naujų lankytinų vietų)

Duomenys1.txt

Kaunas

Ponas Vardenis

Laisvės muziejus2; Laisvės al. 10; 1925; Istorinis; 1 1 1 1 0 0 0; 0; 10

Lietaus muziejus; Kauno al. 3; 1985; Kultūrinis; 1 1 0 1 0 0 1; 1; 25

A; Statulos g. 5; 2016; A; Vardeniui Pavardeniui

Duomenys2.txt

Kaunas

Džėjus Getsbis

Kauno muziejus; Donelaičio g. 2; 1930; Istorinis; 1 1 1 1 1 1 1; 1; 3.90

Laisvės muziejus; Laisvės al. 2; 1915; Istorinis; 1 1 1 1 1 0 0; 1; 5.00

Prezidentūros muziejus; Kauno al. 10; 1923; Istorinis; 1 1 1 1 1 1 1; 0; 10.00

Ne prezidentūros muziejus; Panevėžio al. 5; 2016; Kultūrinis; 1 0 1 1 1 1 1; 1; 15.00

Ne Laisvės muziejus; Panevėžio al. 22; 2016; Kultūrinis; 1 0 1 0 1 1 1; 1; 10

Duomenys3.txt

Kaunas

Džėjus Getsbis

Kaunas; Donelaičio g. 2; 1955; Vardas Pavardė; Janui Libinijui

Prezidentas; Kauno g. 6; 1955; Vardas2 Pavardė2; Jonui Kazlovijui

Ponas; Kauno g. 22; 2013; D; Janui Kazlovijui

A; Statulos g. 5; 2015; A; Vardeniui Pavardeniui

D; Statulos g. 10; 2014; B; Vardeniui2 Pavardeniui2

Super Laisvės muziejus; Panevėžio al. 22; 2012; Kultūrinis; 1 0 1 0 1 1 1; 1; 8

Apskaičiuoti

Muziejų turinčius gidus yra: 6

Seniausia lankoma vieta:

Laisvės muziejus Laisvės al.2 1915 Istorinis 1 1 1 1 0 0 1 5.00

Duomenys1.txt							
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10	
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25	
A	Statulos g. 5	2016	A	Vardeniui Pavardeniui			

Duomenys2.txt							
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90	
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00	
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00	
Ne prezidentūros muziejus	Panevėžio al. 5	2016	Kultūrinis	1 0 1 1 1 1 1	1	15.00	
Ne Laisvės muziejus	Panevėžio al. 22	2016	Kultūrinis	1 0 1 0 1 1 1	1	10	

Duomenys3.txt							
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui			
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui			
Ponas	Kauno g. 22	2013	D	Janui Kazlovijui			
A	Statulos g. 5	2015	A	Vardeniui Pavardeniui			
D	Statulos g. 10	2014	B	Vardeniui2 Pavardeniui2			
Super Laisvės muziejus	Panevėžio al. 22	2012	Kultūrinis	1 0 1 0 1 1 1	1	8	

8 pav. (Ekrano rezultatai)

Pilnas sąrašas.							
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10	
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25	
A	Statulos g. 5	2016	A	Vardeniui Pavardeniui			
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90	
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00	
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00	
Ne prezidentūros muziejus	Panevėžio al. 5	2016	Kultūrinis	1 0 1 1 1 1 1	1	15.00	
Ne Laisvės muziejus	Panevėžio al. 22	2016	Kultūrinis	1 0 1 0 1 1 1	1	10	
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui			
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui			
Ponas	Kauno g. 22	2013	D	Janui Kazlovijui			
A	Statulos g. 5	2015	A	Vardeniui Pavardeniui			
D	Statulos g. 10	2014	B	Vardeniui2 Pavardeniui2			
Super Laisvės muziejus	Panevėžio al. 22	2012	Kultūrinis	1 0 1 0 1 1 1	1	8	

Naujų lankytinų vietų sąrašas.

Tuščias sąrašas.

9 pav. (Ekrano rezultatai)

Sąrašas yra tuščias.

10 pav. (Nauji.csv dokumentas)

Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1	1	25
A	Statulos g. 5	2016	A	Vardeniui Pavardeniui		
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1	1	5.00
Prezidentūros mu	Kauno al. 10	1923	Istorinis	1 1 1 1	0	10.00
Ne prezidentūros	Panevėžio al. 5	2016	Kultūrinis	1 0 1 1	1	15.00
Ne Laisvės muziej	Panevėžio al. 22	2016	Kultūrinis	1 0 1 0	1	10
Kaunas	Donelaičio g. 2	1955	Vardas Pa	Janui Libinijui		
Prezidentas	Kauno g. 6	1955	Vardas2 P	Jonui Kazlovijui		
Ponas	Kauno g. 22	2013	D	Janui Kazlovijui		
A	Statulos g. 5	2015	A	Vardeniui Pavardeniui		
D	Statulos g. 10	2014	B	Vardeniui2 Pavardeniui2		
Super Laisvės muz	Panevėžio al. 22	2012	Kultūrinis	1 0 1 0	1	8

11 pav. (VisosVietos.csv)

Pradiniai duomenys
Dokumento pavadinimas: Duomenys1.txt
Kaunas
Ponas Vardenis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
A	Statulos g. 5	2016	A	Vardeniui Pavardeniui

Dokumento pavadinimas: Duomenys2.txt
Kaunas
Džėjus Getsbis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2016	Kultūrinis	1 0 1 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2016	Kultūrinis	1 0 1 0 1 1 1	1	10

Dokumento pavadinimas: Duomenys3.txt
Kaunas
Džėjus Getsbis

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Super Laisvės muziejus	Panevėžio al. 22	2012	Kultūrinis	1 0 1 0 1 1 1	1	8

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui
Ponas	Kauno g. 22	2013	D	Janui Kazlovijui
A	Statulos g. 5	2015	A	Vardeniui Pavardeniui
D	Statulos g. 10	2014	B	Vardeniui2 Pavardeniui2

Visų lankytinų vietų sąrašas.

Pavadinimas	Adresas	Įkūrimo metai	Tipas	Darbo dienos	Turi gidą?	Bilieto kaina
Laisvės muziejus2	Laisvės al. 10	1925	Istorinis	1 1 1 1 0 0 0	0	10
Lietaus muziejus	Kauno al. 3	1985	Kultūrinis	1 1 0 1 0 0 1	1	25
Kauno muziejus	Donelaičio g. 2	1930	Istorinis	1 1 1 1 1 1 1	1	3.90
Laisvės muziejus	Laisvės al.2	1915	Istorinis	1 1 1 1 1 0 0	1	5.00
Prezidentūros muziejus	Kauno al. 10	1923	Istorinis	1 1 1 1 1 1 1	0	10.00
Ne prezidentūros muziejus	Panevėžio al. 5	2016	Kultūrinis	1 0 1 1 1 1 1	1	15.00
Ne Laisvės muziejus	Panevėžio al. 22	2016	Kultūrinis	1 0 1 0 1 1 1	1	10
Super Laisvės muziejus	Panevėžio al. 22	2012	Kultūrinis	1 0 1 0 1 1 1	1	8

12 pav. (Rezultatai.txt dokumentas)

Pavadinimas	Adresas	Įkūrimo metai	Autorius	Skirta
A	Statulos g. 5	2016	A	Vardeniui Pavardeniui
Kaunas	Donelaičio g. 2	1955	Vardas Pavardė	Janui Libinijui
Prezidentas	Kauno g. 6	1955	Vardas2 Pavardė2	Jonui Kazlovijui
Ponas	Kauno g. 22	2013	D	Janui Kazlovijui
A	Statulos g. 5	2015	A	Vardeniui Pavardeniui
D	Statulos g. 10	2014	B	Vardeniui2 Pavardeniui2

Sąrašas yra tuščias.
Muziejų turinčius gidus yra: 6
Seniausia lankoma vieta: |Laisvės muziejus |Laisvės al.2 | 1915|Istorinis | 1 1 1 1 1 0 0| 1| 5.00|

13 pav. (Rezultatai.txt dokumentas)

Trečias variantas – nėra jokių duomenų dokumentų.

LANKYTINŲ VIETŲ PAIEŠKOS

Nėra duomenų failų!

Apskaičiuoti

14 pav. (Naudotojo vaizdas)

4.8. Dėstytojo pastabos

Testas: 2/3

Gynimas: 5,5/6 (Pakeisti „FileData“ klasės vieną iš savybių – „LinkedList<Location> Locations“ į „private“ atvirumą ir sukurti naują metodą, šiam parametrai pasiekti („public LinkedList<Location> GetLocationsList()“). Sutvarkyta.)

Ataskaita: 1/1

Vienetiniai testai (papildomas balas) – 1.

Galutinis įvertinimas: 9,5 ≈ 10.

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