

Kauno technologijos universitetas

Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Augustinas Jukna IFF-0/3

Studentas

Doc. Renata Burbaitė

Kaunas 2021

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 44

3.4. Klasių diagrama 44

3.5. Programos vartotojo vadovas 44

3.6. Programos tekstas 44

3.7. Pradiniai duomenys ir rezultatai 44

3.8. Dėstytojo pastabos 45

4. Polimorfizmas ir išimčių valdymas (L4) 46

4.1. Darbo užduotis 46

4.2. Grafinės vartotojo sąsajos schema 46

4.3. Sąsajoje panaudotų komponentų keičiamos savybės 46

4.4. Klasių diagrama 46

4.5. Programos vartotojo vadovas 46

4.6. Programos tekstas 46

4.7. Pradiniai duomenys ir rezultatai 46

4.8. Dėstytojo pastabos 47

5. Deklaratyvusis programavimas (L5) 48

5.1. Darbo užduotis 48

5.2. Grafinės vartotojo sąsajos schema 48

5.3. Sąsajoje panaudotų komponentų keičiamos savybės 48

5.4. Klasių diagrama 48

5.5. Programos vartotojo vadovas 48

5.6. Programos tekstas 48

5.7. Pradiniai duomenys ir rezultatai 48

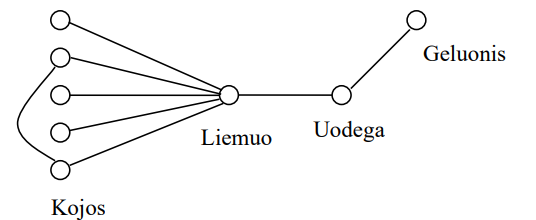
5.8. Dėstytojo pastabos 49

# Rekursija (L1)

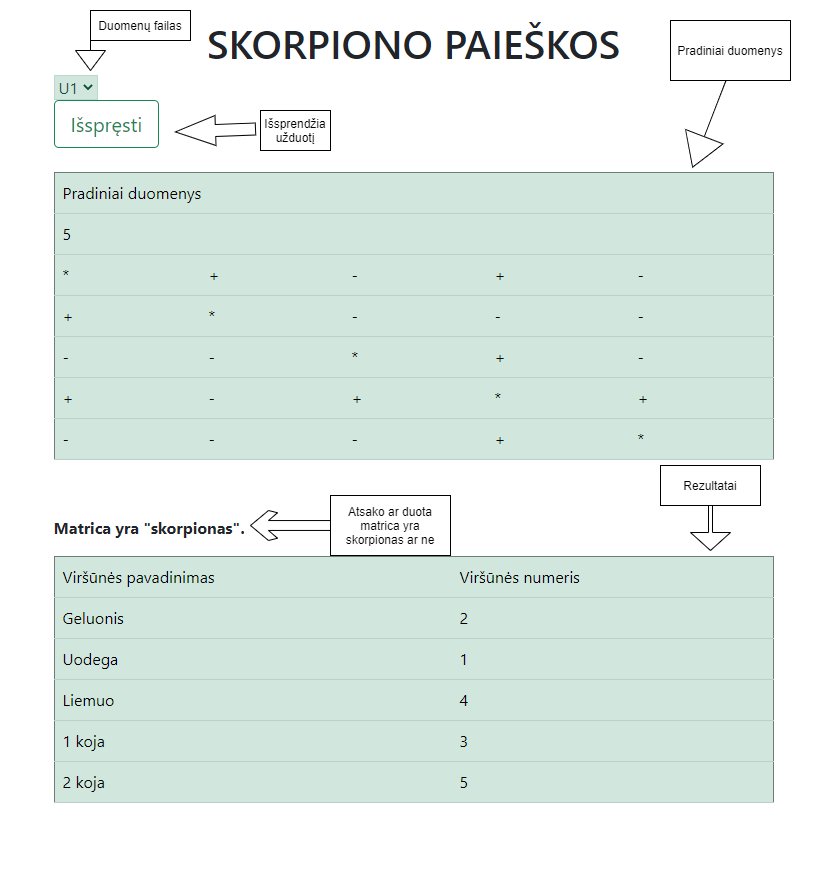
## Darbo užduotis

**LD\_8. Skorpionas.**

Grafas – tai viršūnių ir jas jungiančių briaunų visuma. Tarp dviejų viršūnių gali būti tik viena briauna. Brėžiniuose grafo viršūnės dažnai vaizduojamos mažais apskritimais, o briaunos – linijomis, jungiančiomis šiuos apskritimus. Grafas, turintis n viršūnių yra „skorpionas“, jei yra viena viršūnė (geluonis), sujungta viena briauna su kita viršūne (uodega). Uodega dar viena briauna turi būti sujungta su trečia viršūne (liemeniu). Viršūnė-liemuo jungiama su likusiomis viršūnėmis (kojomis) atskiromis briaunomis. Kai kurios viršūnės-kojos gali būti sujungtos tarpusavyje. Duomenys surašyti tekstiniame faile ’U3.txt’. Pirmoje failo eilutėje yra parašytas sveikasis skaičius n (5 ≤ n ≤ 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“.



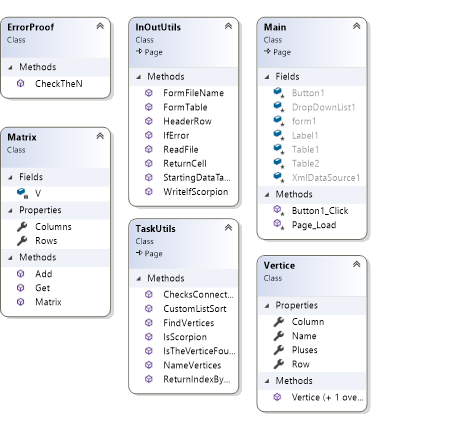
## Grafinės vartotojo sąsajos schema



## 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 |

## Klasių diagrama



## 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.

1. Tokia matrica laikoma skorpionu, kuri:
   1. Turi ne daugiau nei 50 ar ne mažiau nei 5 viršūnes;
   2. turi vieną geluonies viršunę, kuri jungiasi su uodega;
   3. uodegą, kuri jungiasi su geluonimi ir liemeniu;
   4. liemenį, kuris jungiasi su uodega bei viena ar daugiau kojų;
   5. 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.

## Programos tekstas

Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_1\_WebApp

{

public class Matrix

{

private char[,] V;

public int Rows { get; private set; }

public int Columns { get; private set; }

public Matrix(int n)

{

this.V = new char[n, n];

this.Rows = n;

this.Columns = n;

}

/// <summary>

/// Adds a char to a specific place in the container

/// </summary>

/// <param name="i">row to put the object in</param>

/// <param name="j">column to put the object in</param>

/// <param name="character">object</param>

public void Add(int i, int j, char character)

{

this.V[i, j] = character;

}

/// <summary>

/// Gets an object from a specific place

/// </summary>

/// <param name="i">row to take object from</param>

/// <param name="j">column to take object from</param>

/// <returns>the object</returns>

public char Get(int i, int j)

{

return this.V[i, j];

}

}

}

InOutUtils.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.IO;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Lab\_1\_WebApp

{

public class InOutUtils : System.Web.UI.Page

{

/// <summary>

/// Reads data file

/// </summary>

/// <param name="AllLines">Array of all the data</param>

/// <returns>returns a Matrix class object</returns>

public static Matrix ReadFile(string[] AllLines)

{

int n = Int32.Parse(AllLines[0]);

if (!ErrorProof.CheckTheN(n))

{

return null;

}

Matrix allData = new Matrix(n);

for (int i = 0; i < allData.Rows; i++)

{

for (int j = 0; j < allData.Columns; j++)

{

char c = AllLines[i + 1][j];

allData.Add(i, j, c);

}

}

return allData;

}

/// <summary>

/// Forms a file name to fit system's settings

/// </summary>

/// <param name="dropDownList">data file input</param>

/// <returns>returns a fileName</returns>

public static string FormFileName(DropDownList dropDownList)

{

string fileName = "App\_Data/" + dropDownList.SelectedValue + ".txt";

return fileName;

}

/// <summary>

/// Changes label's text based on if a given matrix is a scorpion or not

/// </summary>

/// <param name="label">object to change text</param>

public static void WriteIfScorpion(Label label)

{

label.Text = @"<strong>Matrica yra ""skorpionas"".</strong>";

}

/// <summary>

/// If the matrix is not a scorpion, then changes label's text to fit accordingly

/// </summary>

/// <param name="label">object to change the text</param>

public static void IfError(Label label)

{

label.Text = @"<strong>Ši matrica nėra ""skorpionas"".</strong>";

}

/// <summary>

/// Returns a made cell

/// </summary>

/// <param name="text">cell's text input</param>

/// <returns>a made cell</returns>

public static TableCell ReturnCell(string text)

{

TableCell cell = new TableCell();

cell.Text = text;

return cell;

}

/// <summary>

/// Creates table's header row

/// </summary>

/// <returns>a made header row for table</returns>

public static TableRow HeaderRow()

{

TableRow row = new TableRow();

row.Cells.Add(ReturnCell("Viršūnės pavadinimas"));

row.Cells.Add(ReturnCell("Viršūnės numeris"));

return row;

}

/// <summary>

/// Forms a full table from inputs

/// </summary>

/// <param name="table">Displayed table</param>

/// <param name="vertices">List of all the vertices</param>

public static void FormTable(Table table, List<Vertice> vertices)

{

table.Rows.Add(HeaderRow());

for (int i = 0; i < vertices.Count; i++)

{

TableRow row = new TableRow();

row.Cells.Add(ReturnCell(vertices[i].Name));

row.Cells.Add(ReturnCell((vertices[i].Row + 1).ToString()));

table.Rows.Add(row);

}

}

/// <summary>

/// Fills a table row with empty cells

/// </summary>

/// <param name="row">row to be filled</param>

/// <param name="columns">how many cells to add</param>

public static void FillTableRow(TableRow row, int columns)

{

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

{

row.Cells.Add(ReturnCell(""));

}

}

/// <summary>

/// Creates starting data table for comparison

/// </summary>

/// <param name="table">table to display</param>

/// <param name="matrix">data container</param>

public static void StartingDataTable(Table table, Matrix matrix)

{

TableRow row0 = new TableRow();

TableCell cell = ReturnCell("Pradiniai duomenys");

cell.ColumnSpan = matrix.Columns;

row0.Cells.Add(cell);

TableRow row1 = new TableRow();

row1.Cells.Add(ReturnCell(matrix.Rows.ToString()));

FillTableRow(row1, matrix.Columns - 1);

table.Rows.Add(row0);

table.Rows.Add(row1);

for (int i = 0; i < matrix.Rows; i++)

{

TableRow rowTemp = new TableRow();

for (int j = 0; j < matrix.Columns; j++)

{

rowTemp.Cells.Add(ReturnCell((matrix.Get(i, j)).ToString()));

}

table.Rows.Add(rowTemp);

}

}

/// <summary>

/// Writes lines from vertice's list

/// </summary>

/// <param name="allLines">array of all the lines to write</param>

/// <param name="index">index of line to start writing to</param>

/// <param name="vertices">list of all the vertices</param>

public static void WriteLines(string[] allLines, int index, List<Vertice> vertices)

{

for (int i = 0; i < vertices.Count; i++)

{

allLines[index] = String.Format("Viršūnė: {0, -10} | Numeris: {1}", vertices[i].Name, vertices[i].Row + 1);

index++;

}

}

/// <summary>

/// Creates a string array to hold all the lines

/// </summary>

/// <param name="matrix">data matrix</param>

/// <param name="vertices">all the vertices list</param>

/// <returns>returns a made string array</returns>

public static string[] WriteData(Matrix matrix, List<Vertice> vertices)

{

string[] AllLines = new string[matrix.Rows + vertices.Count + 2];

AllLines[0] = "Pradiniai duomenys";

AllLines[1] = String.Format("n = {0}", matrix.Rows);

for (int i = 2; i <= matrix.Rows + 1; i++)

{

string line = "";

for (int j = 0; j < matrix.Columns; j++)

{

line += matrix.Get(i - 2, j);

}

AllLines[i] = line;

}

return AllLines;

}

}

}

Vertice.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_1\_WebApp

{

public class Vertice

{

public string Name { get; set; }

public int Row { get; set; }

public int Column { get; set; }

public int Pluses { get; set; }

public Vertice(int row, int column)

{

this.Row = row;

this.Column = column;

}

//Empty constructor

public Vertice()

{

this.Row = -1;

this.Column = -1;

}

}

}

TaskUtils.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Lab\_1\_WebApp

{

public class TaskUtils : System.Web.UI.Page

{

/// <summary>

/// Finds all the vertices and lists them (uses recursion)

/// </summary>

/// <param name="matrix">all data container</param>

/// <param name="startRow">starting row</param>

/// <param name="vertices">list of all the vertices</param>

public static void FindVertices(Matrix matrix, int startRow, List<Vertice> vertices)

{

int plusesCount = 0;

Vertice vertice = new Vertice();

for (int j = 0; j < matrix.Columns; j++)

{

if (matrix.Get(startRow, j) == '\*') //if char == '\*', that means, it's i and j, will be the row and the column of the vertice

{

vertice.Row = startRow;

vertice.Column = j;

}

else if (matrix.Get(startRow, j) == '+') //counts how many connections does this vertice has

{

plusesCount++;

}

}

vertice.Pluses = plusesCount;

vertices.Add(vertice);

if (startRow == matrix.Rows - 1) //returns to prevent errors

{

return;

}

else

{

FindVertices(matrix, startRow + 1, vertices);

}

}

/// <summary>

/// Sorts through all the vertices and finds their hierarchy

/// </summary>

/// <param name="matrix">data container</param>

/// <param name="vertices">list of all the vertices</param>

public static void NameVertices(Matrix matrix, List<Vertice> vertices)

{

string sting = "Geluonis", tail = "Uodega", waist = "Liemuo", leg = "koja";

bool flag1 = true, flag2 = true; //keeps method usage in check, so that certain methods would be used once only

int legCount = 0;

for (int i = 0; i < vertices.Count; i++)

{

Vertice vertice = vertices[i];

if (vertice.Pluses == 1 && flag1)

{

for (int j = 0; j < vertices.Count; j++)//starts a new loop to find a suitable vertice that connects to it

{

Vertice vertice2 = vertices[j];

if (ChecksConnection(matrix, vertice.Row, vertice2.Row) && vertice2.Pluses == 2)

{

vertices[j].Name = tail;

vertices[i].Name = sting;

i = 0; //starts a new cycle of loop to not miss any vertices

flag1 = false; //to keep the method from repeating

break;

}

}

}

if (IsTheVerticeFound(vertices, tail) && vertice.Pluses >= 2 && flag2)

{

Vertice vertice2 = vertices[ReturnIndexByName(vertices, tail)];

if (ChecksConnection(matrix, vertice.Row, vertice2.Row))//checks if both of the vertices have a connection ('+')

{

vertices[i].Name = waist;

i = 0;

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

}

}

if (IsTheVerticeFound(vertices, waist) && vertice.Pluses >= 1 && !flag1 && !flag2) //this method will start the last, because both flag1 and flag2 have to be false

{

Vertice vertice2 = vertices[ReturnIndexByName(vertices, waist)];

if (ChecksConnection(matrix, vertice.Row, vertice2.Row))//checks connection

{

legCount++;//counts the legs

vertices[i].Name = legCount + " " + leg;

}

}

}

}

/// <summary>

/// Checks if the vertices connect together

/// </summary>

/// <param name="matrix">data container</param>

/// <param name="vertice1Row">first vertice to check</param>

/// <param name="vertice2Row">second vertice to check</param>

/// <returns>returns a true or false statement</returns>

public static bool ChecksConnection(Matrix matrix, int vertice1Row, int vertice2Row)

{

if (matrix.Get(vertice2Row, vertice1Row) == '+')

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Checks if the vertice is already in the list and named

/// </summary>

/// <param name="vertices">list of all the vertices</param>

/// <param name="name">name of the needed vertice</param>

/// <returns>a true or false statement</returns>

public static bool IsTheVerticeFound(List<Vertice> vertices, string name)

{

foreach (Vertice vertice in vertices)

{

if (vertice.Name == name)

{

return true;

}

}

return false;

}

/// <summary>

/// Returns a vertice's index by name

/// </summary>

/// <param name="vertices">list of all the vertices</param>

/// <param name="name">name of the vertice</param>

/// <returns>true or false statement</returns>

public static int ReturnIndexByName(List<Vertice> vertices, string name)

{

for (int i = 0; i < vertices.Count; i++)

{

if (vertices[i].Name == name)

{

return i;

}

}

return -1;

}

/// <summary>

/// Sorts the list in a custom manner

/// </summary>

/// <param name="vertices">list of all the vertices</param>

public static void CustomListSort(List<Vertice> vertices)

{

string sting = "Geluonis", tail = "Uodega", waist = "Liemuo";

Vertice temp = new Vertice();

for (int i = 0; i < vertices.Count; i++)

{

if (vertices[i].Name == sting)

{

temp = vertices[0];

vertices[0] = vertices[i];

vertices[i] = temp;

}

if (vertices[i].Name == tail)

{

temp = vertices[1];

vertices[1] = vertices[i];

vertices[i] = temp;

}

if (vertices[i].Name == waist)

{

temp = vertices[2];

vertices[2] = vertices[i];

vertices[i] = temp;

}

}

}

/// <summary>

/// Checks if the matrix is scorpion

/// </summary>

/// <param name="vertices">list of all the vertices</param>

/// <returns>a true or false statement</returns>

public static bool IsScorpion(List<Vertice> vertices)

{

foreach (Vertice vertice in vertices)

{

if (vertice.Name == null)

{

return false;

}

}

return true;

}

}

}

ErrorProof.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Lab\_1\_WebApp

{

public class ErrorProof

{

/// <summary>

/// Checks if there are too many inputs

/// </summary>

/// <param name="n">the amount of inputs</param>

/// <returns>a true or false statement</returns>

public static bool CheckTheN(int n)

{

if (n > 50 || n < 5)

{

return false;

}

else

{

return true;

}

}

}

}

Main.aspx.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.IO;

namespace Lab\_1\_WebApp

{

public partial class Main : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void Button1\_Click(object sender, EventArgs e)

{

string[] AllLines = File.ReadAllLines(Server.MapPath(InOutUtils.FormFileName(DropDownList1)));//inputs data

Matrix scorpionMatrix = InOutUtils.ReadFile(AllLines);

if (scorpionMatrix == null) //checks if the data is correct

{

Label1.Text = "<strong>Neteisingi duomenys!</strong>";

File.WriteAllText(Server.MapPath("App\_Data/Rezultatai.txt"), "Neteisingi duomenys.");

return;

}

InOutUtils.StartingDataTable(Table2, scorpionMatrix);

List<Vertice> AllVertices = new List<Vertice>();

TaskUtils.FindVertices(scorpionMatrix, 0, AllVertices); //first lists all the vertices

TaskUtils.NameVertices(scorpionMatrix, AllVertices); //names them

TaskUtils.CustomListSort(AllVertices); //sorts them

bool isScorpion = TaskUtils.IsScorpion(AllVertices); //checks if given data is a scorpion

if (!isScorpion)

{

InOutUtils.IfError(Label1);

string[] WrittenLines = InOutUtils.WriteData(scorpionMatrix, AllVertices);

File.WriteAllLines(Server.MapPath("App\_Data/Rezultatai.txt"), WrittenLines);

File.AppendAllText(Server.MapPath("App\_Data/Rezultatai.txt"), "Tai nėra skorpionas.");

return;

}

else

{

string[] WrittenLines = InOutUtils.WriteData(scorpionMatrix, AllVertices);

InOutUtils.WriteLines(WrittenLines, scorpionMatrix.Rows + 2, AllVertices);

File.WriteAllLines(Server.MapPath("App\_Data/Rezultatai.txt"), WrittenLines);

InOutUtils.WriteIfScorpion(Label1);

InOutUtils.FormTable(Table1, AllVertices);

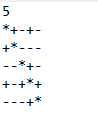
}

}

}

}

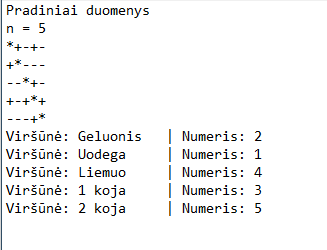
## Pradiniai duomenys ir rezultatai

1. 

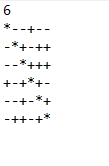
U1.txt



Rezultatai



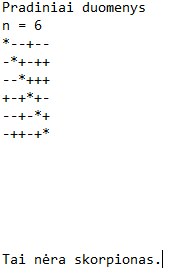
Rezultatai.txt

1. 

U2.txt

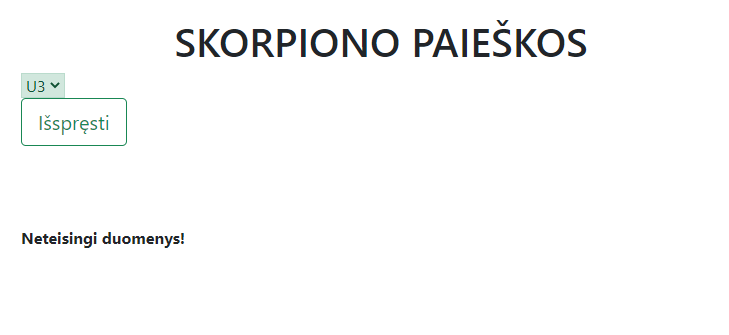


Rezultatai



Rezultatai.txt

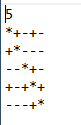
1. U3.txt = 51



Rezultatai



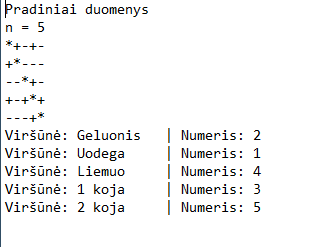
Rezultatai.txt

1. 

U4.txt



Rezultatai



Rezultatai.txt

## Dėstytojo pastabos

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

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

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

(Sutvarkyta)

# Dinaminis atminties valdymas (L2)

## Darbo užduotis

**LD\_8. Maršrutai.**

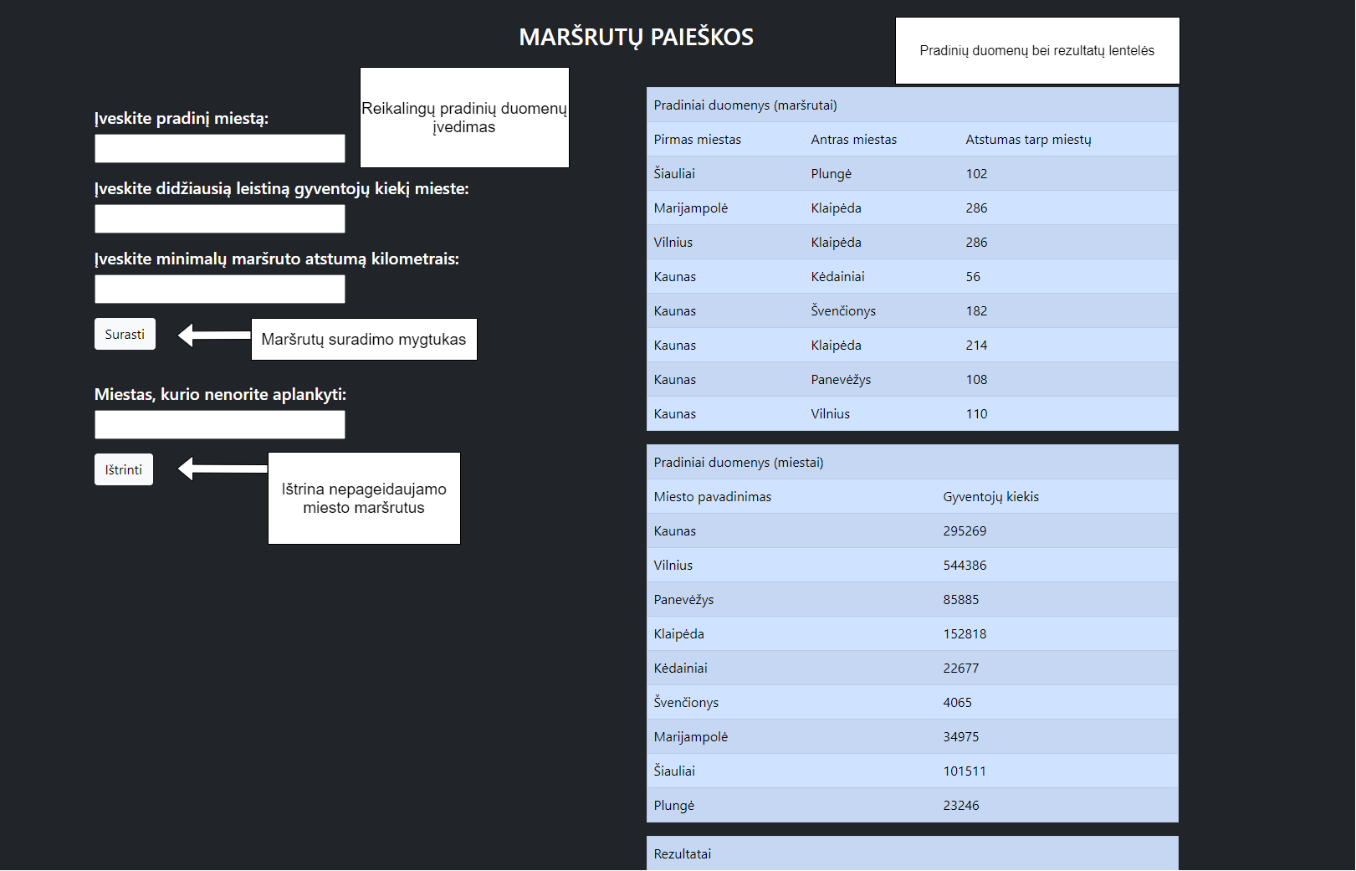
Turizmo agentūra rengia kelionę po Lietuvą iš nurodyto miesto. Išvykus iš vieno miesto galima nukeliauti į bet kurį kitą miestą. Tarp miestų gali būti daugiau kaip vienas kelionės maršrutas. Kelionės metu tas pats miestas gali būti aplankytas tik vieną kartą ir galima lankyti tik tuos miestus, kuriuose 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.

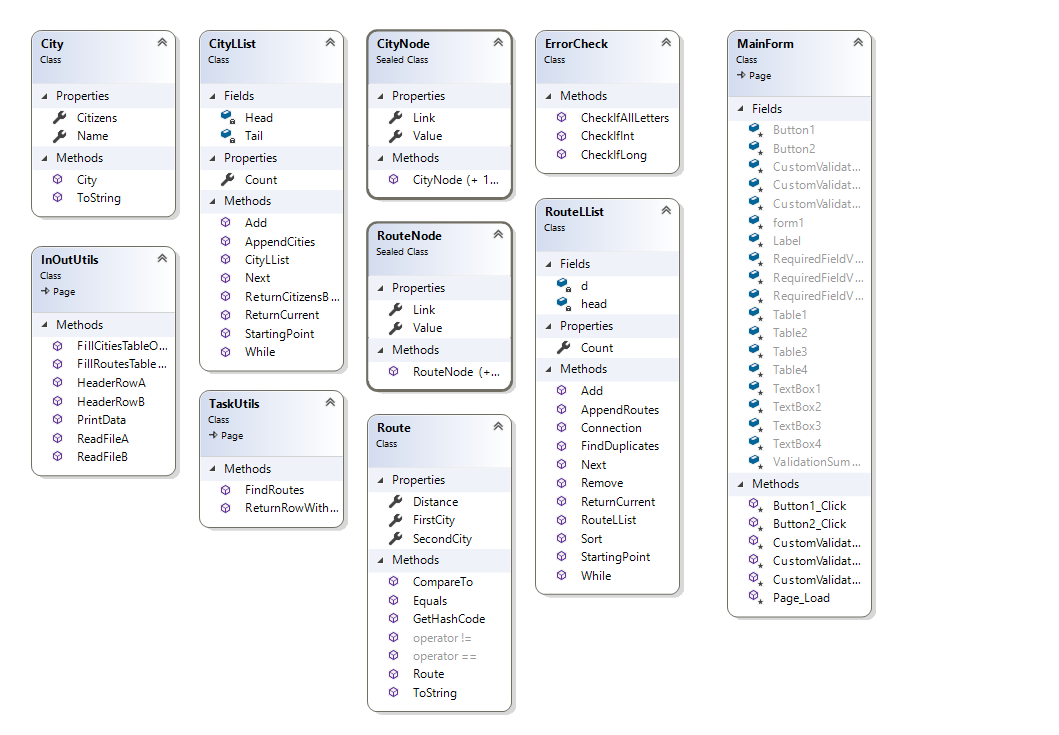
## Grafinės vartotojo sąsajos schema



## 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 |

## Klasių diagrama



## Programos vartotojo vadovas

* Atsidarius puslapiui, vartotoją pasitinką kelios įvestys bei mygtukas „Surasti“.
* Norint, jog programa pradėtu 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.

## 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;

Label.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;

}

}

}

}

## 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

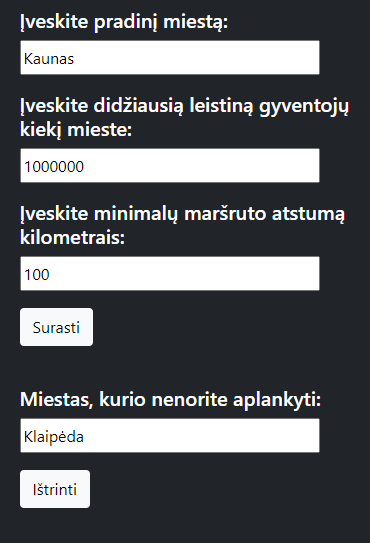
Švenčionys;4065

Marijampolė;34975

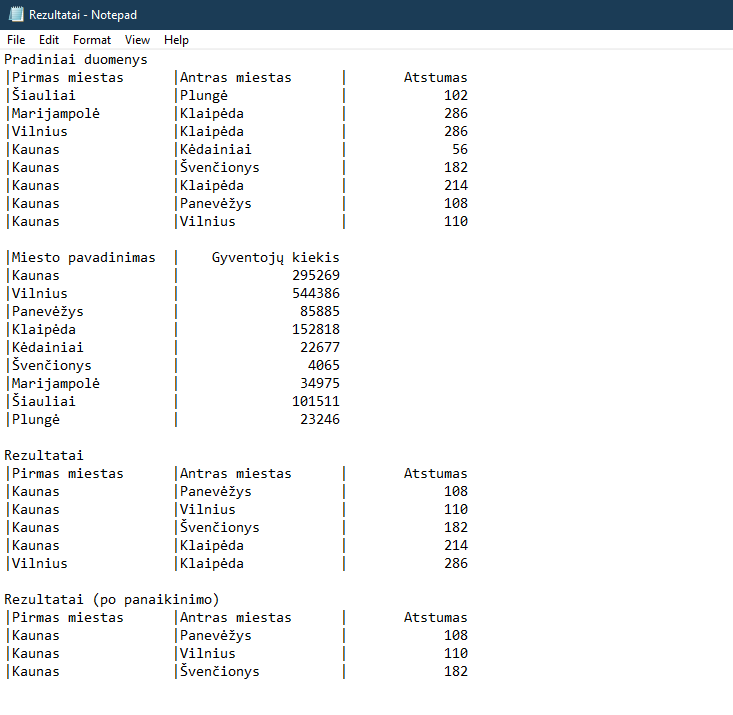
Šiauliai;101511

Plungė;23246

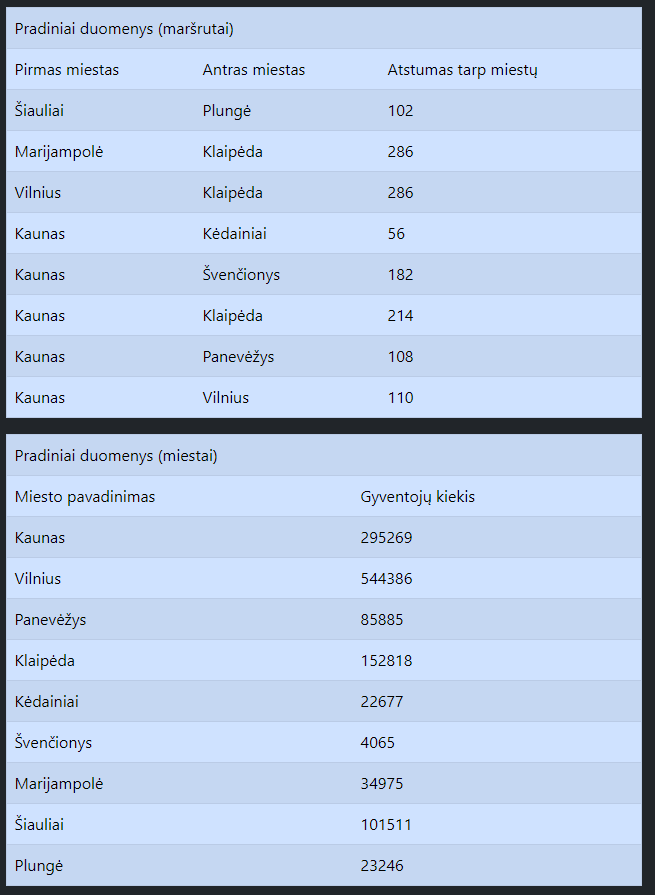
Ekrano įvestys:

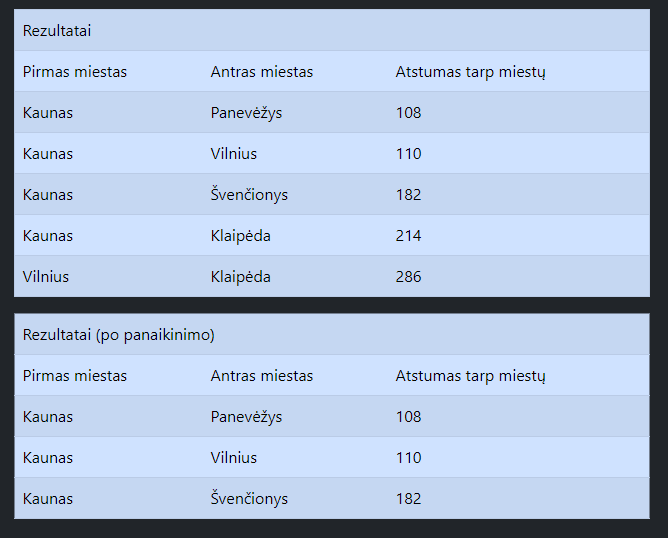


Rezultatai.txt



Ekrano rezultatai





2 variantas)

U8a.txt

Vilnius;Klaipėda;286

Marijampolė;Klaipėda;286

Šiauliai;Plungė;102

U8b.txt

Vilnius;100000

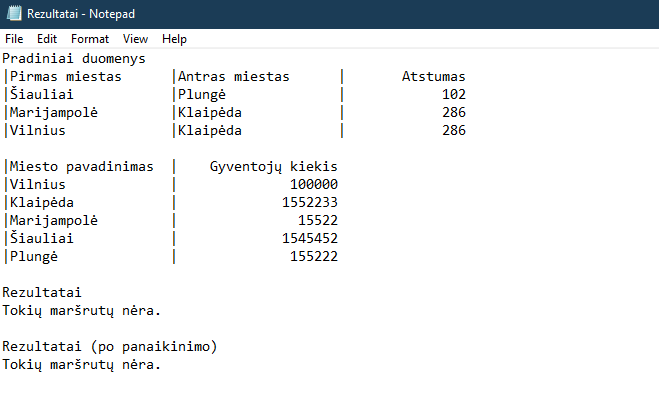
Klaipėda;1552233

Marijampolė;15522

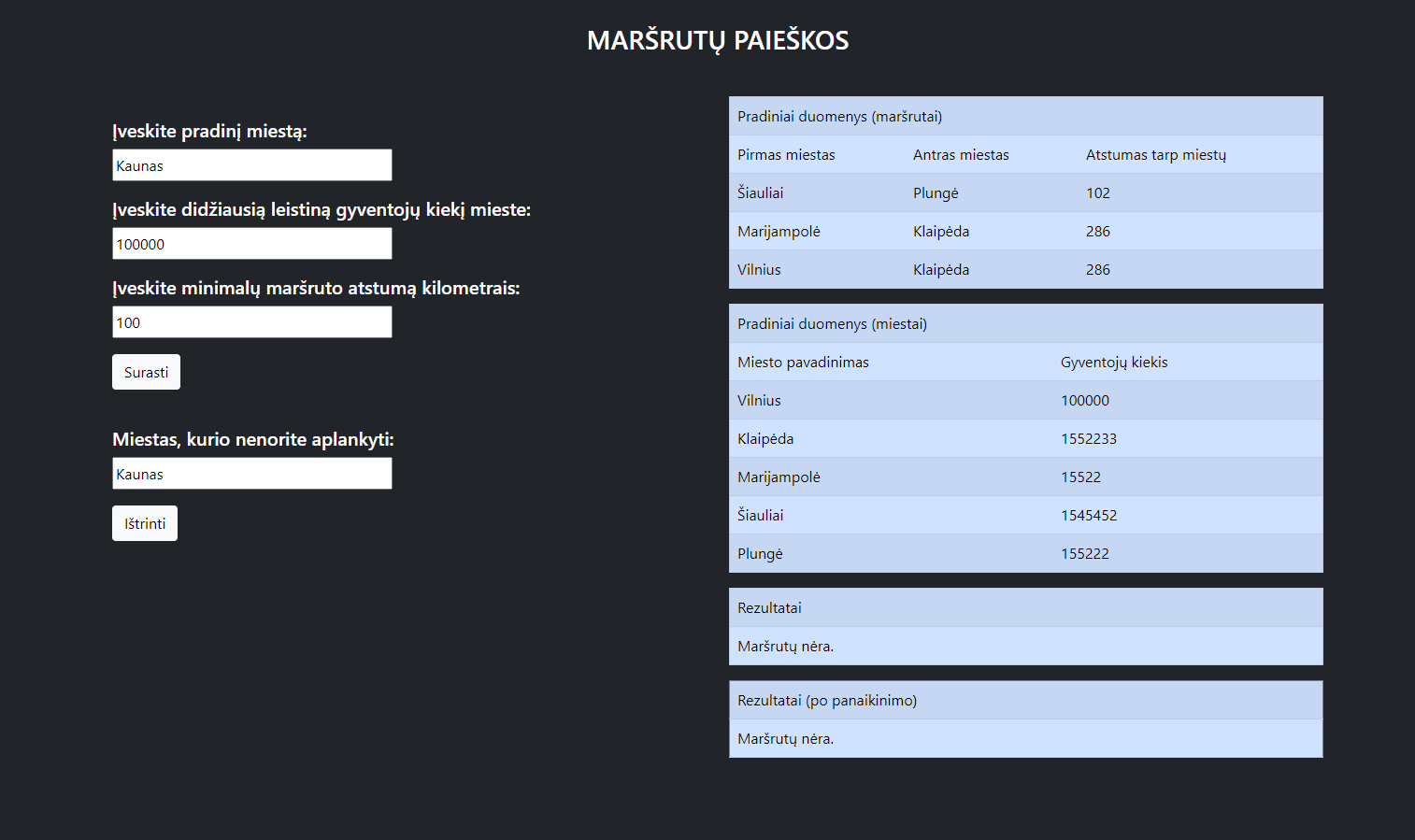
Šiauliai;1545452

Plungė;155222

Rezultatai.txt



Ekrano įvestys/rezultatai



3 variantas)

Nėra vieno ar abiejų duomenų failų.



## Dėstytojo pastabos

# Bendrinės klasės ir testavimas (L3)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos

# Polimorfizmas ir išimčių valdymas (L4)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos

# Deklaratyvusis programavimas (L5)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos