KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

DISKREČIOSIOS STRUKTŪROS (P170B008)

KURSINIS DARBAS

Užduoties Nr. C17

Atliko: IFF-0/5 gr. studentas *Povilas Paškevičius*

Priėmė: dėst. *Audrius Nečiūnas*

KAUNAS

2021

Turinys

1. Užduotis (C lygis, Nd17) 3
2. Užduoties analizė 3
3. Programos tekstas 4
4. Testavimo pavyzdžiai 11
5. Išvados 16
6. Literatūros sąrašas 16

1. Užduotis (C lygis, Nd17)

Grafas rodo, kas gyvenvietėje yra pažįstami. Gandas sklinda nuo „autoriaus“ kitiems. Kiekviename žingsnyje kiekvienas žinantis gandą gali jį perduoti vienam iš pažįstamų. Parašyti programą, kuri nustatytų, kelių žingsnių gandui reikės (mažiausiai), kad pasiektų visus gyventojus.

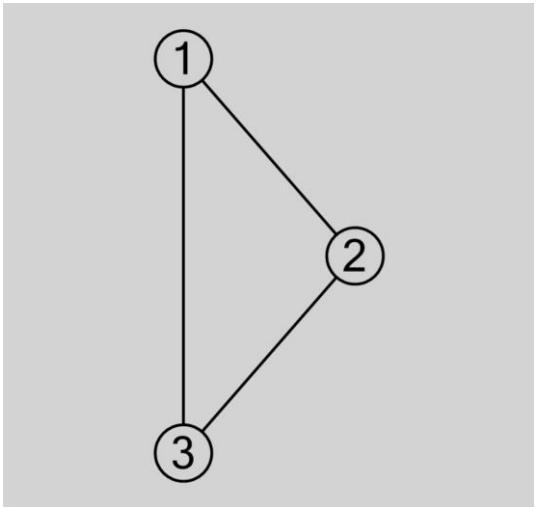
2. Užduoties analizė

Sprendžiant šį uždavinį bus laikoma, kad grafas yra neorientuotasis (nes kai grafas orientuotasis, gandas gali po jį pasklisti tik tuo atveju, jei jis yra stipriai jungus) bei jungusis (nes kai grafas nejungus, gandui neįmanoma pasiekti visų jo viršūnių).

Programa gaus grafą, aprašytą gretimumo struktūra. Vartotojui bus galima pasirinkti, kurioje grafo viršūnėje susiformavo gandas. Atlikus skaičiavimus programa į ekraną išves visus galimus trumpiausius gando sklidimo kelius.

Jeigu gandas susiformavo grafe, kuris yra sudarytas iš 1 viršūnės, tuomet galima teigti, kad gandas jau pasklido ir jokiu skaičiavimų neatlikinėti. Kitais atvejais uždavinys bus sprendžiamas taip:

1. Įvestas grafas išsaugomas gretimumo struktūroje (Adjs). Sukuriamas gando sklidimo keliams formuoti ir saugoti skirtas masyvas (Solve), kuriame pradedami formuoti keliai nuo pasirinktos viršūnės;
2. Kol į nagrinėjamą kelią nėra įtraukta tiek viršūnių kiek yra grafo gretimumo struktūroje, tol iš kiekvienos to kelio viršūnės einama į visas galimas viršūnes ir pagal tai suformuojami nauji keliai;
3. Jei nagrinėjamas kelias turi tiek pat viršūnių kiek ir grafas, vadinasi tai yra trumpiausias gando sklidimo kelias, o visi už jo Solve masyve esantys keliai yra netrumpesni už gautąjį;
4. Iš Solve masyvo išrenkami tik tie keliai, kurie yra tokio pačio ilgio (kelio ilgis – žingsnių skaičius per kurį gandas pasklinda po grafą), kaip prieš tai rastas trumpiausias kelias ir susideda iš tiek pat viršūnių, kiek jų turi pateiktas grafas.



1 pav. Grafas su trimis viršūnėmis

1 pav. pateiktame grafe, gandą skleidžiant iš bet kurios viršūnės, bus surasti 2 galimi keliai, kurie abu yra trumpiausi. Toks rezultatas bus gautas, nes pateiktas algoritmas blogiausiu atveju peržiūrės visus galimus gando sklidimo kelius iš pasirinktos viršūnės ir išrinks tik tuos kelius kurie yra trumpiausi.

Anksčiau aprašytą algoritmą galima realizuoti keliomis programavimo kalbomis, palygintomis 1 lentelėje.

1 lentelė. Programavimo kalbų palyginimas

C#	MATLAB
Populiari programavimo kalba, patogi sintaksė, lengvai prieinama bei išsami dokumentacija, bibliotekų įvairovė bei lengvas jų diegimas	Patogus darbas su matricomis, paprasta grafų vizualizacija, lengvai prieinama dokumentacija

Ši užduotis bus realizuojama C# programavimo kalba, nes buvo rasta biblioteka, su kuria yra paprasta atvaizduoti grafus. Be to, C# programavimo kalba buvo mokoma programuoti 1 kurse.

Taip pat reikėtų pasirinkti kokiu būdu būtų patogiausia aprašyti grafą. Grafo aprašymo būdų palyginimas pateiktas 2 lentelėje.

2 lentelė. Grafo aprašymo būdų palyginimas

Kriterijus	Gretimumo matrica	Incidencijų matrica	Briaunų matrica	Gretimumo struktūra	Masyvais L ir lst
Viršūnių, gretimų duotajai, radimo paprastumas	Sunku	Sunku	Vidutiniškai sunku	Paprasta	Vidutiniškai sunku

Kadangi šiame uždavinyje dažnai reikės atlikti, duotajai viršūnei gretimų, viršūnių peržiūrą, dėl to buvo pasirinka aprašyti grafą Gretimumo struktūra.

3. Programos tekstas

GraphInfo.cs

```
using Microsoft.Msagl.Drawing;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GandasMain
{
    public class GraphInfo
    {
        public Graph solution { get; private set; }
        public string steps { get; private set; }
        public GraphInfo(Graph startingGraph, Path path)
```

```
        {
            solution = startingGraph;
            ColorGraph(path);
            FormSteps(path);
        }
        private void ColorGraph(Path path)
        {
            for(int i = 0; i < path.EdgesCount; i++)
            {
                foreach(var one in solution.Edges)
                {
                    if(one.SourceNode.Id == (path.GetEdge(i).SourceNode +
1).ToString() && one.TargetNode.Id == (path.GetEdge(i).TargetNode +
1).ToString() || one.SourceNode.Id == (path.GetEdge(i).TargetNode +
1).ToString() && one.TargetNode.Id == (path.GetEdge(i).SourceNode +
1).ToString())
                    {
                        one.SourceNode.Attr.FillColor = Color.Blue;
                        one.TargetNode.Attr.FillColor = Color.Blue;
                        one.Attr.Color = Color.Blue;
                        break;
                    }
                }
            }
            solution.FindNode((path.GetVertex(0) + 1).ToString()).Attr.FillColor
= Color.Red;
        }
        private void FormSteps(Path path)
        {
            int start = 0;
            for(int i = 0; i < path.Steps; i++)
            {
                steps += (i + 1) + " žingsnis:\n";
                for(int j = start; j < path.GetHistory(i); j++)
                {
                    steps += (path.GetEdge(j).SourceNode + 1) + " -> " +
(path.GetEdge(j).TargetNode + 1) + "\n";
                }
                start = path.GetHistory(i);
            }
            steps = steps.Trim();
        }
    }
}
```

GraphSolver.cs

```
using Microsoft.Msagl.Drawing;
using Microsoft.Msagl.Layout.Layered;
using System;
using System.Collections.Generic;
using System.Linq;

namespace GandasMain
{
    public class GraphSolver
    {
        private List<List<int>> Adjs;
        public GraphSolver(List<List<int>> adjs)
        {
            Adjs = adjs;
```

```
        }
        public Graph GetStartingGraph()
        {
            Graph graph = new Graph();
            graph.Directed = false;
            var settings = new SugiyamaLayoutSettings();
            settings.NodeSeparation = 50;
            settings.LayerSeparation = 50;
            settings.EdgeRoutingSettings.EdgeRoutingMode =
Microsoft.Msagl.Core.Routing.EdgeRoutingMode.StraightLine;
            graph.LayoutAlgorithmSettings = settings;
            graph.Attr.BackgroundColor = Color.LightGray;
            if (Adjs.Count == 0)
            {
                Node node = new Node("1");
                node.Attr.Shape = Shape.Circle;
                graph.AddNode(node);
            }
            else
            {
                for (int i = 0; i < Adjs.Count; i++)
                {
                    for (int j = 0; j < Adjs[i].Count; j++)
                    {
                        if (i + 1 < Adjs[i][j])
                        {
                            Edge one = graph.AddEdge((i + 1).ToString(),
Adjs[i][j].ToString());
                            one.Attr.ArrowheadAtTarget = ArrowStyle.None;
                            if (one.SourceNode.Attr.Shape != Shape.Circle)
                            {
                                one.SourceNode.Attr.Shape = Shape.Circle;
                            }
                            one.TargetNode.Attr.Shape = Shape.Circle;
                        }
                    }
                }
            }
            return graph;
        }
        public List<GraphInfo> SpreadRumors(int Start)
        {
            if (Adjs.Count == 0)
            {
                List<GraphInfo> Solution = new List<GraphInfo>();
                Path path = new Path();
                path.AddVertex(Start - 1);
                GraphInfo one = new GraphInfo(GetStartingGraph(), path);
                Solution.Add(one);
                return Solution;
            }
            else
            {
                List<Path> Solve = new List<Path>();
                Path temp = new Path();
                temp.AddVertex(Start - 1);
                Solve.Add(temp);
                int i = 0;
                while (Solve[i].VerticesCount < Adjs.Count)
                {
                    List<Path> pathVariations = new List<Path>();
```

```
                    for (int j = 0; j < Solve[i].VerticesCount; j++)
                    {
                        GeneratePaths(Solve[i].GetVertex(j), ref pathVariations,
Solve[i]);
                    }
                    AppendPaths(Solve[i], pathVariations, Solve);
                    i++;
                }
                List<Path> best = SelectBestPaths(i, Solve);
                return GenerateSolutionGraph(best);
            }
        }
        private void GeneratePaths(int from, ref List<Path> pathVariations, Path
rest)
        {
            if (pathVariations.Count == 0)
            {
                for (int i = 0; i < Adjs[from].Count; i++)
                {
                    if (!rest.ContainsVertex(Adjs[from][i] - 1))
                    {
                        Path temp = new Path();
                        temp.AddVertex(Adjs[from][i] - 1);
                        temp.AddEdge(new SimpleEdge(from, Adjs[from][i] - 1));
                        pathVariations.Add(temp);
                    }
                }
            }
            else
            {
                List<Path> newVariations = new List<Path>();
                for (int i = 0; i < pathVariations.Count; i++)
                {
                    bool log = true;
                    for (int j = 0; j < Adjs[from].Count; j++)
                    {
                        if (!pathVariations[i].ContainsVertex(Adjs[from][j] - 1)
&& !rest.ContainsVertex(Adjs[from][j] - 1))
                        {
                            log = false;
                            Path pathCpy = new Path(pathVariations[i]);
                            pathCpy.AddVertex(Adjs[from][j] - 1);
                            pathCpy.AddEdge(new SimpleEdge(from, Adjs[from][j] -
1));
                            newVariations.Add(pathCpy);
                        }
                    }
                    if (log) newVariations.Add(pathVariations[i]);
                }
                pathVariations = newVariations;
            }
        }
        private void AppendPaths(Path rest, List<Path> pathVariations,
List<Path> solve)
        {
            for (int i = 0; i < pathVariations.Count; i++)
            {
                Path Appended = new Path(rest);
                Appended.AppendPath(pathVariations[i]);
                solve.Add(Appended);
            }
```

```
        }
        private List<Path> SelectBestPaths(int index, List<Path> solve)
        {
            int criteria = solve[index].Steps;
            List<Path> best = new List<Path>();
            for (int i = index; i < solve.Count; i++)
            {
                if (solve[i].Steps == criteria && solve[i].VerticesCount ==
Adjs.Count)
                {
                    best.Add(solve[i]);
                }
            }
            return best;
        }
        private List<GraphInfo> GenerateSolutionGraph(List<Path> best)
        {
            List<GraphInfo> solutions = new List<GraphInfo>();
            for (int i = 0; i < best.Count; i++)
            {
                GraphInfo solution = new GraphInfo(GetStartingGraph(), best[i]);
                solutions.Add(solution);
            }
            return solutions;
        }
    }

}
```

InOut.cs

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

namespace GandasMain
{
    public static class InOut
    {
        public static GraphSolver ReadGraph(string FileName)
        {
            List<List<int>> Adjs = new List<List<int>>();
            string[] lines = File.ReadAllLines(FileName);
            string message = "";
            int ErrorCount = 0;
            for(int i = 0; i < lines.Length; i++)
            {
                string[] parts = lines[i].Split(new string[] { " " },
StringSplitOptions.RemoveEmptyEntries);
                List<int> vertex = new List<int>();
                for(int j = 0; j < parts.Length;j++)
                {
                    try
                    {
                        int value = int.Parse(parts[j]);
                        if (value > 0)
                        {
                            vertex.Add(value);
```

```
                        }
                        else
                        {
                            ErrorCount++;
                            int line = i + 1;
                            message = message + line + ", ";
                            break;
                        }
                    }
                    catch(FormatException)
                    {
                        ErrorCount++;
                        int line = i + 1;
                        message = message + line + ", ";
                        break;
                    }
                }
                Adjs.Add(vertex);
            }
            if(!String.IsNullOrEmpty(message))
            {
                if(ErrorCount == 1)
                {
                    message = "Eilutės nr: " + message.Remove(message.Length -
2);
                    throw new FormatException(message);
                }
                else
                {
                    message = "Eilučių nr: " + message.Remove(message.Length -
2);
                    throw new FormatException(message);
                }
            }
            GraphSolver solver = new GraphSolver(Adjs);
            return solver;
        }
    }

}
```

Path.cs

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

namespace GandasMain
{
    public class Path
    {
        private List<int> Vertices;
        private List<SimpleEdge> Edges;
        public int VerticesCount
        {
            get
            {
                return Vertices.Count;
            }
```

```
}
public int EdgesCount
{
    get
    {
        return Edges.Count;
    }
}
private List<int> History;
public int Steps
{
    get
    {
        return History.Count;
    }
}
public Path(Path path)
{
    Vertices = new List<int>();
    Edges = new List<SimpleEdge>();
    History = new List<int>();
    for (int i = 0; i < path.Vertices.Count; i++)
    {
        Vertices.Add(path.Vertices[i]);
    }
    for (int i = 0; i < path.Edges.Count; i++)
    {
        Edges.Add(path.Edges[i]);
    }
    for(int i = 0; i < path.History.Count; i++)
    {
        History.Add(path.History[i]);
    }
}
public Path()
{
    Vertices = new List<int>();
    Edges = new List<SimpleEdge>();
    History = new List<int>();
}
public bool ContainsVertex(int element)
{
    return Vertices.Contains(element);
}
public void AddEdge(SimpleEdge edge)
{
    Edges.Add(edge);
}
public SimpleEdge GetEdge(int index)
{
    return Edges[index];
}
public void AddVertex(int vertex)
{
    Vertices.Add(vertex);
}
public int GetVertex(int index)
{
    return Vertices[index];
}
public int GetHistory(int index)
```

```
        {
            return History[index];
        }
        public void AppendPath(Path other)
        {
            for (int i = 0; i < other.VerticesCount; i++)
            {
                Vertices.Add(other.GetVertex(i));
            }
            for (int i = 0; i < other.EdgesCount; i++)
            {
                Edges.Add(other.GetEdge(i));
            }
            History.Add(Edges.Count);
        }
    }

}
```

SimpleEdge.cs

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

namespace GandasMain
{
    public class SimpleEdge
    {
        public int SourceNode { get; set; }
        public int TargetNode { get; set; }
        public SimpleEdge(int S, int T)
        {
            SourceNode = S;
            TargetNode = T;
        }
    }

}
```

4. Testavimo pavyzdžiai

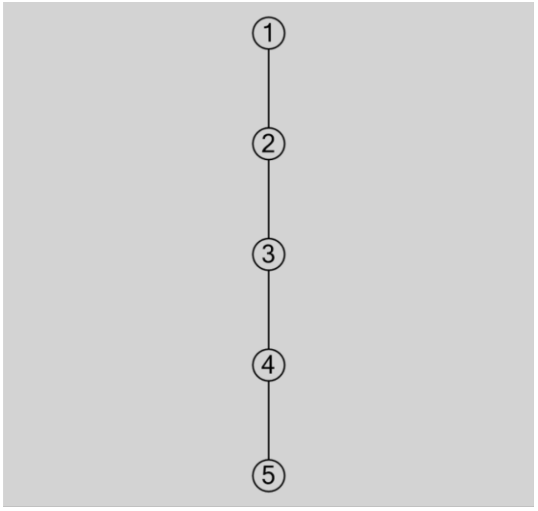
Buvo panaudoti trys testavimo pavyzdžiai:

Pirmas Testas

Faile Grafas.txt pateikta tokia gretimumo struktūra:

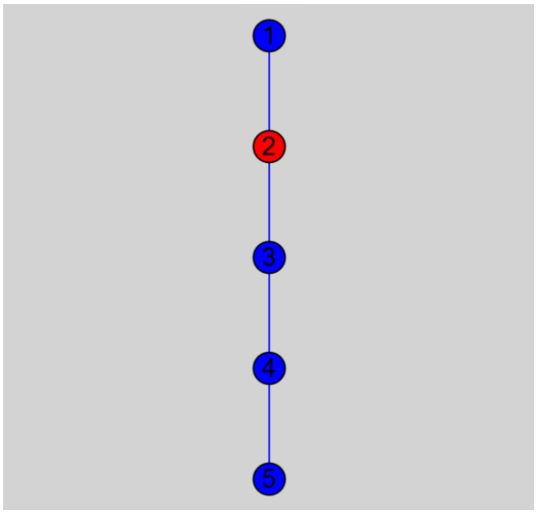
```
2
1 3
2 4
3 5
4
```

Pirmojo pavyzdžio grafas pavaizduotas 2 pav.:



2 pav. Iš faile Grafas.txt pateiktos gretimumo struktūros gautas grafas

Jei 2 viršūnė pasirenkama, kaip pradinė viršūnė, gaunamas rezultatas pateiktas 3 pav.:



3 pav. Pirmo testo rezultatas

Gando sklidimo kelias žingsniais:

1. 2 → 3
2. 2 → 1; 3 → 4
3. 4 → 5

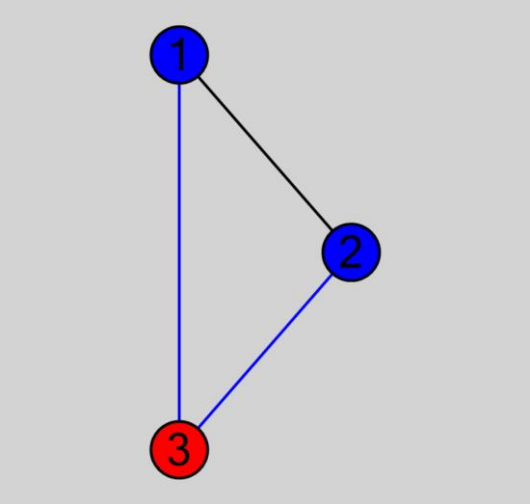
Antras testas

Faile Grafas.txt pateikta tokia gretimumo struktūra:

```
2 3
1 3
2 1
```

Antrojo pavyzdžio grafas pavaizduotas 1 pav.

Jei 3 viršūnė pasirenkama, kaip pradinė viršūnė, gaunamas rezultatas pateiktas 4 pav.:



4 pav. Antro testo rezultatas

Šiuo atveju gaunami 2 trumpiausi keliai, kurie atrodo lygiai taip pat. Pirmasis kelias:

1. 3 → 2
2. 3 → 1

Antrasis kelias:

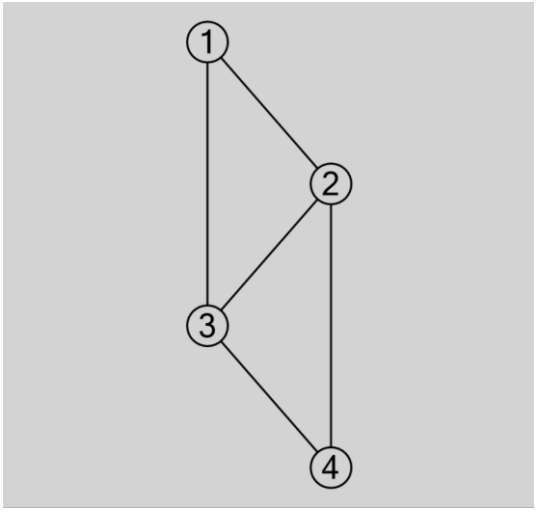
1. 3 → 1
2. 3 → 2

Trečias testas

Faile Grafas.txt pateikta tokia gretimumo struktūra:

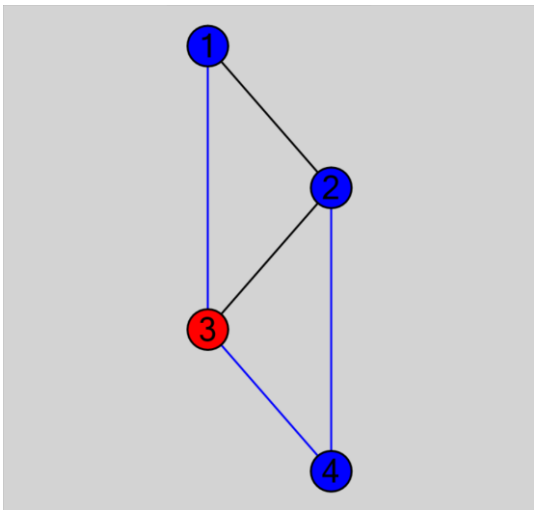
```
2 3
1 3 4
4 1 2
2 3
```

Trečiojo pavyzdžio grafas pavaizduotas 5 pav.:

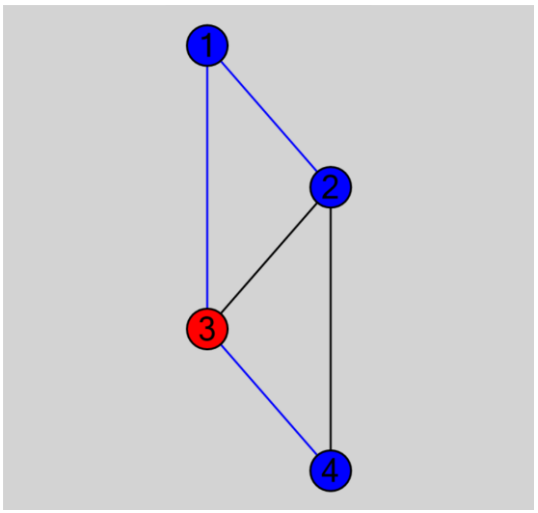


5 pav. . Iš faile Grafas.txt pateiktos gretimumo struktūros gautas grafas

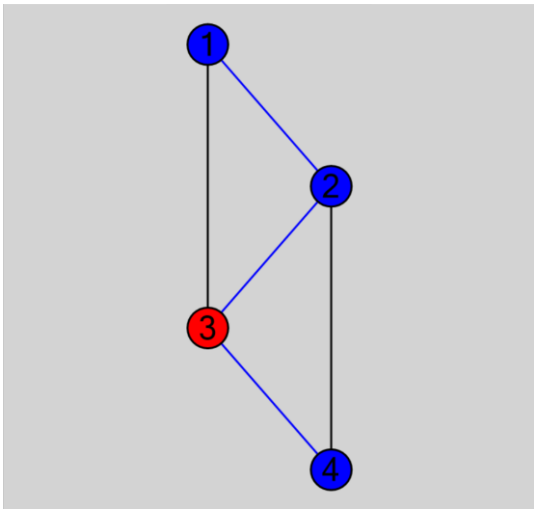
Jei 3 viršūnė pasirenkama, kaip pradinė viršūnė, gaunami rezultatai pateikti 6 – 10 pav.:



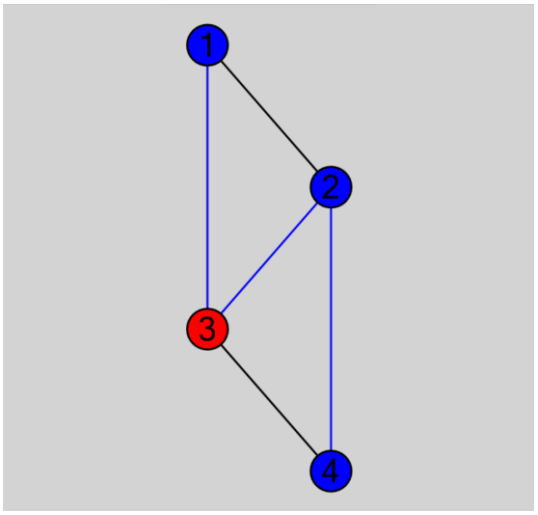
6 pav. Pirmas trečio testo rezultatas



7 pav. Antras trečio testo rezultatas



8 pav. Trečias trečio testo rezultatas



9 pav. Ketvirtas trečio testo rezultatas

Šiuo atveju gaunami 4 trumpiausi keliai. Pirmas kelias:

1. $3 \rightarrow 4$
2. $3 \rightarrow 1;\ 4 \rightarrow 2$

Antras kelias:

1. $3 \rightarrow 1$
2. $3 \rightarrow 4;\ 1 \rightarrow 2$

Trečias kelias:

1. $3 \rightarrow 2$
2. $3 \rightarrow 4;\ 2 \rightarrow 1$

Ketvirtas kelias:

1. $3 \rightarrow 2$

2. $3 \rightarrow 1; 2 \rightarrow 4$

5. Išvados

Programa veikia teisingai.

Šios užduoties sprendimo algoritmas yra pakankamai lėtas, nes norint taisyklingai išspręsti šį uždavinį reikia rasti visus galimus gando sklidimo kelius. Tai reiškia, kad šis uždavinys yra NP-pilnas.

6. Literatūros sąrašas

1. „Diskrečiųjų struktūrų“ modulis „Moodle“ aplinkoje https://moodle.ktu.edu/course/view.php?id=39 (žiūrėta 2021-11-20)
2. „Microsoft Automatic Graph Layout“ dokumentacija https://github.com/microsoft/automatic-graph-layout (žiūrėta 2021-11-19)
3. C# programavimo kalbos dokumentacija https://docs.microsoft.com/en-us/dotnet/csharp/ (žiūrėta 2021-11-19)