Algoritmica grafelor laboratorul 2
Tema: obtinerea valorilor unor drumuri (valoare minima intre doua noduri date folosind alg. Yen).

Enunt

Sa se conceapa un algoritm care primeste ca parametru de intrare o lista reprezentand toate varfurile unui graf, iar pentru fiecare element din lista se memoreaza o lista de perechi $\mathbb{N}x\mathbb{R}^+$ reprezentand lista predecesorilor acelui varf (partea real pozitiva fiind valoare arcului de la predecesr la varf) si doua valori, prima reprezentand varful sursa si cea de a doua valoare reprezentand varful destinatie. Algoritmul va returna un numar real reprezentand valoarea minima dintre toate valorile drumurilor care pornesc din varful sursa si se termina in varful destinatie. Graful nu contine cicluri de valori negative.

Dezvoltarea algoritmului

Algoritmul nu poate fi impartit in subprobleme deoarece reprezinta un algoritm de calcul al valorilor drumurilor de la un varf la altul.

Descrierea algoritmului

```
functie ValoareMinima(listaPredecesorilor, sursa, destinatie)
    λ_i ← l_sursa, i, i = 0..(listaPredecesorilor.NumarElemente)

    repeta
        k ← 0
        pentru j = 0..(listaPredecesorilor.NumarElemente), j ≠ sursa executa
            pentru i = 1..(listaPredecesorilor.NumarElemente), i ∈ Γ⁻_j executa
                daca λ_i + l_i, j < λ_j atunci
                    λ_j ← λ_i + l_i, j
                    k ← 1
                sfarsit daca
            sfarsit pentru
        sfarsit pentru
        pentru j = (listaPredecesorilor.NumarElemente)..0, j ≠ sursa executa
            pentru i = 1..(listaPredecesorilor.NumarElemente), i ∈ Γ⁻_j executa
                daca λ_i + l_i, j < λ_j atunci
                    λ_j ← λ_i + l_i, j
                    k ← 1
                sfarsit daca
            sfarsit pentru
        sfarsit pentru
    panacand k = 0

    ValoareMinima ← λ_destinatie
sfarsit functie
```

Demostrarea corectitudinii

Algoritmul de calcul este preluat din curs (algoritmul lui Yen) si usor modificat. In curs s-a dat un algoritm care calculeaza valoarea minima a drumurilor care pornesc din varful 1 si se termina in varful j. Adaptarea este relativ simpla deoarece initializarea lui λ_i se facea cu valorile drumurilor care pornesc din 1 si se termina in i cu conditia ca intre cele doua varfuri sa existe un arc de la 1 la i. In cazul in care nu exista se lua valoarea plus infinit iar in cazul in care i = 1 se lua valoarea 0.

Adaptarea aici a fost abstractizarea valorii 1 cu o valoare data. Daca inainte se calcula de la varful 1

la i acum se calculeaza de la sursa la i folosind exact aceasi metoda, daca exista arc de la varful sursa la varful i se considera valoarea arcului de la sursa la i, daca sursa = i se considera valoarea 0, altfel se considera valoarea plus infinit.

In cadrul calculului, pentru algoritmul dat la curs) j parcurge toate varfurile mai putin varful 1 (cel din care se porneste). Si aici s-a aplicat aceasi metoda, sa abstractizat varful de la care se porneste (nu mai este varful 1 este un varful sursa). Asadar j parcurge toate varfurile mai putin varful sursa. Functia returneaza valoarea $\lambda_{destinatie}$ deoarece pe acea pozitie se afla valoarea minima dintre valorile drumurilor care pornesc din sursa si se termina in destinatie.

In cazul in care sursa este inlocuit cu 1 se obtine un algoritm echivlent cu cel dat la curs.

Cod sursa (C#)

```
static public double Yen(IReadOnlyList<IReadOnlyDictionary<int, double>> predecessorsLists,
                         int sourcePeek, int destinationPeek)
{
    if (predecessorsLists != null)
        if (predecessorsLists.Count > 0)
            if (0 <= sourcePeek
                && sourcePeek < predecessorsLists.Count)
                if (0 <= destinationPeek
                    && destinationPeek < predecessorsLists.Count)
                {
                    List<int> allPeeksExceptSource = new List<int>();
                    double[] minimumRoadValues = new double[predecessorsLists.Count];

                    for (int peek = 0; peek < minimumRoadValues.Length; peek++)
                        if (peek == sourcePeek)
                            minimumRoadValues[peek] = 0;
                        else
                        {
                            allPeeksExceptSource.Add(peek);
                            if (!predecessorsLists[peek].TryGetValue(sourcePeek,
                                out minimumRoadValues[peek]))
                                minimumRoadValues[peek] = double.PositiveInfinity;
                        }

                    bool foundASmallerRoadValue;

                    do
                    {
                        foundASmallerRoadValue = false;

                        for (int peekIndex = 0;
                             peekIndex < allPeeksExceptSource.Count;
                             peekIndex++)
                            foreach (KeyValuePair<int, double> predecessor
                                     in predecessorsLists[allPeeksExceptSource[peekIndex]])
                                if (minimumRoadValues[predecessor.Key] + predecessor.Value
                                    < minimumRoadValues[allPeeksExceptSource[peekIndex]])
                                {
                                    minimumRoadValues[allPeeksExceptSource[peekIndex]]
                                        = minimumRoadValues[predecessor.Key]
                                          + predecessor.Value;
                                    foundASmallerRoadValue = true;
                                }
```

```
                    for (int peekIndex = allPeeksExceptSource.Count - 1;
                         peekIndex >= 0;
                         peekIndex--)
                        foreach (KeyValuePair<int, double> predecessor
                                 in predecessorsLists[allPeeksExceptSource[peekIndex]])
                            if (minimumRoadValues[predecessor.Key] + predecessor.Value
                                < minimumRoadValues[allPeeksExceptSource[peekIndex]])
                            {
                                minimumRoadValues[allPeeksExceptSource[peekIndex]]
                                    = minimumRoadValues[predecessor.Key]
                                      + predecessor.Value;
                                foundASmallerRoadValue = true;
                            }
                } while (foundASmallerRoadValue);

                return minimumRoadValues[destinationPeek];
            }
            else
                throw new ArgumentOutOfRangeException("destinationPeek");
        else
            throw new ArgumentOutOfRangeException("sourcePeek");
    else
        throw new ArgumentException("The graph is empty!");
else
    throw new ArgumentNullException("predecessorsLists");
}
```

Date de test

```
[TestClass]
public class MinRoadsTests
{
    [TestClass]
    public class YenTests
    {
        static private readonly IReadOnlyList<IReadOnlyDictionary<int, double>>
            _onePeekGraph = new[]
        {
            new KeyValuePair<int, double>[0]
                .ToDictionary(pair => pair.Key, pair => pair.Value)
        };
        static private readonly IReadOnlyList<IReadOnlyDictionary<int, double>>
            _twoPeekGraph = new[]
        {
            new KeyValuePair<int, double>[0]
                .ToDictionary(pair => pair.Key, pair => pair.Value),

            new [] { new KeyValuePair<int, double>(0, 1) }
                .ToDictionary(pair => pair.Key, pair => pair.Value)
        };
        static private readonly IReadOnlyList<IReadOnlyDictionary<int, double>>
            _fourPeekGraphWithTwoRoutesEqualInLength = new[]
        {
            new KeyValuePair<int, double>[0]
                .ToDictionary(pair => pair.Key, pair => pair.Value),

            new [] { new KeyValuePair<int, double>(0, 1) }
                .ToDictionary(pair => pair.Key, pair => pair.Value),
```

```
        new [] { new KeyValuePair<int, double>(0, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(2, 1),
                 new KeyValuePair<int, double>(3, 2) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),
    };
    static private readonly IReadOnlyList<IReadOnlyDictionary<int, double>>
        _fivePeekGraphWithACycle = new[]
    {
        new KeyValuePair<int, double>[0]
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(0, 1),
                 new KeyValuePair<int, double>(3, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(1, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(2, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(1, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value)
    };
    static private readonly IReadOnlyList<IReadOnlyDictionary<int, double>>
        _fivePeekGraphWithARouteLongerThanTheOtherButCheaper = new[]
    {
        new KeyValuePair<int, double>[0]
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(0, 2) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(1, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(2, 1),
                 new KeyValuePair<int, double>(4, 10) }
            .ToDictionary(pair => pair.Key, pair => pair.Value),

        new [] { new KeyValuePair<int, double>(0, 1) }
            .ToDictionary(pair => pair.Key, pair => pair.Value)
    };

    [TestMethod]
    public void TestWhenTheGraphHasOnlyOnePeek()
    {
        Assert.AreEqual(0, MinRoads.Yen(_onePeekGraph, 0, 0));
    }

    [TestMethod]
    public void TestWhenTheGraphHasTwoPeeksAndTheSourceAndDestinationAreTheSame()
    {
        Assert.AreEqual(0, MinRoads.Yen(_twoPeekGraph, 0, 0));
        Assert.AreEqual(0, MinRoads.Yen(_twoPeekGraph, 1, 1));
    }
```

```
        [TestMethod]
        public void TestWhenTheGraphHasTwoPeeksAndThereIsNoRoadFromSourceToDestination()
        {
            Assert.IsTrue(double.IsPositiveInfinity(MinRoads.Yen(_twoPeekGraph, 1, 0)));
        }

        [TestMethod]
        public void TestWhenTheGraphHasTwoPeeksAndThereIsARoadFromSourceToDestination()
        {
            Assert.AreEqual(1, MinRoads.Yen(_twoPeekGraph, 0, 1));
        }

        [TestMethod]
        public void TestWhenTheGrapHasTwoRoutesEqualInLengthButOneIsCheaper()
        {
            Assert.AreEqual(2, MinRoads.Yen(_fourPeekGraphWithTwoRoutesEqualInLength,
                            0, 3));
        }

        [TestMethod]
        public void TestWhenTheGrapHasACircuit()
        {
            Assert.AreEqual(2, MinRoads.Yen(_fivePeekGraphWithACycle, 0, 4));
        }

        [TestMethod]
        public void TestWhenTheGrapHasARouteLongerThanTheOtherButCheaper()
        {
            Assert.AreEqual(4,
                MinRoads.Yen(_fivePeekGraphWithARouteLongerThanTheOtherButCheaper, 0, 3));
        }
    }
}
```

Aici se afla grafele intr-o reprezentare grafica pentru a vizualiza mai usor datele de test. Acestea apar in aceasi ordine in care apar declarate mai sus.