Algoritmica grafelor laboratorul 3
Tema: obinerea matricei distantelor minime.

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 numere naturale reprezentand lista predecesorilor acelui varf. Algoritmul va returna o matrice de numere reale reprezentand matricea distantelor minime ale grafului dat.

Dezvoltarea algoritmului

Pentru rezolvarea problemei se va da fiecarui arc valoarea 1, in acest caz valoarea unui drum coincide cu distanta acestuia. Apoi pentru fiecare doua varfuri se determina valoarea minima pentru drumurile dintre acestea (de exemplu folosind algoritmul lui Yen) si se completeaza in matricea distantelor in mod corespunzator.

Descrierea algoritmului

```
functie MatriceaDistantelorMinime(listaPredecesorilor)
    listaPredecesorilor2 ← AtaseazaValoare(listaPredecesorilor, 1)

    pentru i ← 0, listaPredecesorilor2.NumarElemente executa
        pentru j ← 0, listaPredecesorilor2.NumarElemente executa
            matriceaDistantelor[i, j] ← ValoareMinima(listaPredecesorilor2,i,j).
        sfarsit pentru
    sfarsit pentru

    MatriceaDistantelorMinime ← matriceaDistantelor
sfarsit functie

functie AtaseazaValoare(lista, valoare)
    rezultat ← CreazaLista(lista.NumarElemente)

    pentru i ← 0, lista.NumarElemente executa
        rezutlat.Adauga(CreazaPereche(lista[i], valoare)
    sfarsit pentru

    AtaseazaValoare ← rezultat
sfarsit functie
```

```
functie ValoareMinima(listaPredecesorilor, sursa, destinatie)
    λi ← lsursa, 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 λi + li, j < λj atunci
                    λj ← λi + li, 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 λi + li, j < λj atunci
                    λj ← λi + li, j
                    k ← 1
                sfarsit daca
            sfarsit pentru
        sfarsit pentru
    panacand k = 0

    ValoareMinima ← λdestinatie
sfarsit functie
```

Demostrarea corectitudinii

Algoitmul este corect deoarece fiecarui arc i se asociaza valoarea 1 dupa care pentru toate perechile de varfuri se aplica algoritmul lui Yen (din laboratorul 2) pentru a calcula valoarea minima dintre valorile drumurilor care pornesc dintr-un varf si se termina in celalalt. Matricea distantelor este completata corespunzator.

Demonstratia corectitudinii pentru functia ValoareMinima (luata din laboratorul 2):

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#)

```
using AlgoritmicaGrafelor.Laborator2.DrumuriMinime;

namespace AlgoritmicaGrafelor.Laborator3.MatriceaDistantelor
{
    public static class DistanceMatrix
    {
        static public double[,] MinimumRoadsDistance(IReadOnlyList<IReadOnlyList<int>>
predecessorsLists)
        {
            if (predecessorsLists != null)
            {
                double[,] distanceMatrix = new double[predecessorsLists.Count,
                                                      predecessorsLists.Count];
                List<IReadOnlyDictionary<int, double>> predecessorsListsWithValuesOfOne
                    = new List<IReadOnlyDictionary<int, double>>();

                foreach (IReadOnlyList<int> predecessorsList in predecessorsLists)
                    predecessorsListsWithValuesOfOne.Add(_AddValueForEach(predecessorsList,
                                                         1d));

                for (int startPeek = 0; startPeek < predecessorsLists.Count; startPeek++)
                    for (int endPeek = 0; endPeek < predecessorsLists.Count; endPeek++)
                        if (startPeek == endPeek)
                            distanceMatrix[startPeek, endPeek] = 0;
                        else
                            distanceMatrix[startPeek, endPeek]
                                = MinRoads.Yen(predecessorsListsWithValuesOfOne,
                                               startPeek,
                                               endPeek);

                return distanceMatrix;
            }
            else
                throw new ArgumentNullException("predecessorsLists");
        }

        static private IReadOnlyDictionary<int, double> _AddValueForEach(IReadOnlyList<int>
list, double value)
        {
            if (list != null)
            {
                IDictionary<int, double> result = new Dictionary<int, double>();

                foreach (int item in list)
                    result.Add(item, value);

                return new ReadOnlyDictionary<int, double>(result);
            }
            else
                throw new ArgumentNullException("list");
        }
    }
}
```

Date de test

```
namespace AlgoritmicaGrafelor.Laborator2.DrumuriMinime.Tests
```

```
{
    [TestClass]
    public class DistanceMatrixTests
    {
        [TestClass]
        public class MinimumRoadsDistance
        {
            static private readonly IReadOnlyList<IReadOnlyList<int>> _onePeekGraph = new[]
            {
                new List<int>{ }
            };
            static private readonly IReadOnlyList<IReadOnlyList<int>> _twoPeekGraph = new[]
            {
                new List<int>{ },
                new List<int>{ 0 }
            };
            static private readonly IReadOnlyList<IReadOnlyList<int>>
_fourPeekGraphWithTwoRoadsEqualInLengthForSameExtremities = new[]
            {
                new List<int>{ },
                new List<int>{ 0 },
                new List<int>{ 0 },
                new List<int>{ 1, 2 }
            };
            static private readonly IReadOnlyList<IReadOnlyList<int>>
_fivePeekGraphWithACycle = new[]
            {
                new List<int>{ },
                new List<int>{ 0, 3 },
                new List<int>{ 1 },
                new List<int>{ 2 },
                new List<int>{ 1 }
            };
            static private readonly IReadOnlyList<IReadOnlyList<int>>
_fivePeekGraphWithARoadLongerThanTheOtheForSameExtremities = new[]
            {
                new List<int>{ },
                new List<int>{ 0 },
                new List<int>{ 1 },
                new List<int>{ 2, 4 },
                new List<int>{ 0 }
            };

            [TestMethod]
            public void TestForOnePeekGraph()
            {
                double[,] expectedResultMatrix = new[,]{
                    { 0d } };
                double[,] actualResultMatrix
                    = DistanceMatrix.MinimumRoadsDistance(_onePeekGraph);

                _AreEqual(expectedResultMatrix, actualResultMatrix, 1, 1);
            }

            [TestMethod]
            public void TestForTwoPeekGraphWithOneRoadFromZeroToOne()
            {
                double[,] expectedResultMatrix = new[,] {
                    { 0,                        1 },
                    { double.PositiveInfinity, 0 } };
```

```
            double[,] actualResultMatrix
               = DistanceMatrix.MinimumRoadsDistance(_twoPeekGraph);

            _AreEqual(expectedResultMatrix, actualResultMatrix, 2, 2);
        }

        [TestMethod]
        public void TestForFourPeekGraphHavingTwoRoadsOfSameLength()
        {
            double[,] expectedResultMatrix = new[,] {
                { 0,                        1,                        1,
2 },
                { double.PositiveInfinity, 0,
double.PositiveInfinity, 1 },
                { double.PositiveInfinity, double.PositiveInfinity, 0,
1 },
                { double.PositiveInfinity, double.PositiveInfinity,
double.PositiveInfinity, 0 } };
            double[,] actualResultMatrix
                =
DistanceMatrix.MinimumRoadsDistance(_fourPeekGraphWithTwoRoadsEqualInLengthForSameExtremitie
s);

            _AreEqual(expectedResultMatrix, actualResultMatrix, 3, 3);
        }

        [TestMethod]
        public void TestForFivePeekGrapthWithCycle()
        {
            double[,] expectedResultMatrix = new[,] {
                { 0,                        1,                        2,
3,                        2 },
                { double.PositiveInfinity, 0,                        1,
2,                        1 },
                { double.PositiveInfinity, 2,                        0,
1,                        3 },
                { double.PositiveInfinity, 1,                        2,
0,                        2 },
                { double.PositiveInfinity, double.PositiveInfinity,
double.PositiveInfinity, double.PositiveInfinity, 0 } };
            double[,] actualResultMatrix
                = DistanceMatrix.MinimumRoadsDistance(_fivePeekGraphWithACycle);

            _AreEqual(expectedResultMatrix, actualResultMatrix, 4, 4);

        }

        [TestMethod]
        public void
TestForFivePeekGraphContainingTwoRoadsOfDifferentLengthsForSameExtremities()
        {
            double[,] expectedResultMatrix = new[,] {
                { 0,                        1,                        2,
2, 1 },
                { double.PositiveInfinity, 0,                        1,
2, double.PositiveInfinity },
                { double.PositiveInfinity, double.PositiveInfinity, 0,
1, double.PositiveInfinity },
                { double.PositiveInfinity, double.PositiveInfinity,
double.PositiveInfinity, 0, double.PositiveInfinity },
```

```
                    { double.PositiveInfinity, double.PositiveInfinity,
double.PositiveInfinity, 1, 0 } };
                double[,] actualResultMatrix
                    =
DistanceMatrix.MinimumRoadsDistance(_fivePeekGraphWithARoadLongerThanTheOtheForSameExtremiti
es);

                _AreEqual(expectedResultMatrix, actualResultMatrix, 5, 5);
            }

            private void _AreEqual(double[,] expectedResultMatrix, double[,]
actualResultMatrix, int lineCount, int columnCount)
            {
                for (int line = 0; line < lineCount; line++)
                    for (int column = 0; column < columnCount; column++)
                        if (expectedResultMatrix[line, column]
                                != actualResultMatrix[line, column])
                            Assert.Fail("Not equal!");
            }
        }
    }
}
```

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.