Algoritmica grafelor laboratorul 5
Tema: arbori partiali de valoare minima.

Enunt

Sa se conceapa un algoritm care primeste ca parametru de intrare o lista muchiilor unui graf (triplete NxNxR). Graful este conex. Algoritmul va returna un arbore partial de valoare minima unde valoarea grafului este suma valorilor muchiilor din arbore. Arborele returnat este reprezentat sub aceasi forma ca si parameterul de intrare. Graful contine cel putin un varf.

Dezvoltarea algoritmului

Problema nu este suficient de complexa pentru a fi impartita in subprobleme deoarece exista deja algoritmi care avand un graf conex determina subarborele partial de valoare minima (de exemplu algoritmul lui Prim).

Descrierea algoritmului

```
functie DeterminaSubarborePartial(multimeaMuchilor)
    varfuri ← multimeaMuchilor.Varfuri
    varfuriVizitate ← CreazaLista()
    varfuriVizitate ← varfuri.Primul
    muchiiVizitate ← CreazaLista()
    cattimp varfuri ≠ varfuriVizitate executa
        pentru varf ∈ varfuriVizitate executa
            muchii_i ← min{m ∈ multimeaMuchilor.Adiacenta(varf) |
                        m.Varfuri\{varf} ∪ varfuriVizitate ≠ varfuriVizitate}
        sfarsit pentru
        muchie ← min{muchii}
        varfuriVizitate ← varfuriVizitate ∪ muchie.Varfuri
        muchiiVizitate ← muchiiVizitate ∪ muchie
    sfarsit cattimp
    DeterminaSubarborePartial ← muchiiVizitate
sfarsit functie
```

Demostrarea corectitudinii

Algoritmul este preluat din curs, se considera ca fiind corect.

Cod sursa (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace AlgoritmicaGrafelor.Laborator5.Subarbori
{
    public static class SubarborePartial
    {
        static public IReadOnlyCollection<Tuple<int, int, double>>
Prim(IReadOnlyCollection<Tuple<int, int, double>> multimeaMuchiilor)
        {
            if (multimeaMuchiilor == null)
                throw new ArgumentNullException("muchii");
            if (multimeaMuchiilor.Count == 0)
                return new Tuple<int, int, double>[0];
            IReadOnlyCollection<int> varfuri = multimeaMuchiilor.SelectMany(muchie => new[]
```

```
{ muchie.Item1, muchie.Item2 })
                                                        .Distinct()
                                                        .ToList();
            ISet<int> varfuriVizitate = new HashSet<int> { varfuri.First() };
            ISet<Tuple<int, int, double>> muchiiVizitate = new HashSet<Tuple<int, int,
double>>();
            while (varfuriVizitate.Count != varfuri.Count)
            {
                IList<Tuple<int, int, double>> muchii = new List<Tuple<int, int, double>>();
                foreach (int varf in varfuriVizitate)
                {
                    Tuple<int, int, double> muchieAdiacentaDeValoareMinima =
                        multimeaMuchiilor.Where(m => (m.Item1 == varf || m.Item2 == varf)
                                                    && varfuriVizitate.Contains(m.Item1) !
= varfuriVizitate.Contains(m.Item2))
                                         .OrderBy(m => m.Item3)
                                         .FirstOrDefault();
                    if (muchieAdiacentaDeValoareMinima != null)
                        muchii.Add(muchieAdiacentaDeValoareMinima);
                }
                Tuple<int, int, double> muchie = muchii.OrderBy(m => m.Item3).First();
                varfuriVizitate.Add(muchie.Item1);
                varfuriVizitate.Add(muchie.Item2);
                muchiiVizitate.Add(muchie);
            }
            return muchiiVizitate.ToList();
        }
    }
}
```

Date de test

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace AlgoritmicaGrafelor.Laborator5.Subarbori.Tests
{
    [TestClass]
    public class PrimTests
    {
        static private readonly IReadOnlyCollection<Tuple<int, int, double>> _onePeekGraph =
            new Tuple<int, int, double>[0];
        static private readonly IReadOnlyCollection<Tuple<int, int, double>> _twoPeekGraph =
            new[] { Tuple.Create(0, 1, 1d) };
        static private readonly IReadOnlyCollection<Tuple<int, int, double>> _threePeekGraph
=
            new[] { Tuple.Create(0, 1, 1d),
                    Tuple.Create(0, 2, 2d),
                    Tuple.Create(1, 2, 1d) };
        static private readonly IReadOnlyCollection<Tuple<int, int, double>> _fourPeekGraph
=
            new[] { Tuple.Create(1, 2, 4d),
                    Tuple.Create(0, 2, 2d),
                    Tuple.Create(0, 3, 3d),
                    Tuple.Create(3, 2, 1d) };
        [TestMethod]
        public void TestForOnePeekGraph()
        {
```

```
            IReadOnlyCollection<Tuple<int, int, double>> result =
SubarborePartial.Prim(_onePeekGraph);
            Assert.IsNotNull(result);
            Assert.AreEqual(0, result.Count);
        }
        [TestMethod]
        public void TestForTwoPeekGraph()
        {
            IReadOnlyCollection<Tuple<int, int, double>> result =
SubarborePartial.Prim(_twoPeekGraph);
            Assert.IsNotNull(result);
            Assert.AreEqual(1, result.Count);
            Assert.IsTrue(result.SequenceEqual(_twoPeekGraph));
        }
        [TestMethod]
        public void TestForThreePeekGraph()
        {
            IReadOnlyCollection<Tuple<int, int, double>> result =
SubarborePartial.Prim(_threePeekGraph);
            Assert.IsNotNull(result);
            Assert.AreEqual(2, result.Count);
            Assert.IsTrue(result.OrderBy(tuple => tuple.Item2)
                                .OrderBy(tuple => tuple.Item1)
                                .SequenceEqual(new []
                                {
                                    Tuple.Create(0, 1, 1d),
                                    Tuple.Create(1, 2, 1d)
                                }));
        }
        [TestMethod]
        public void TestForFourPeekGraph()
        {
            IReadOnlyCollection<Tuple<int, int, double>> result =
SubarborePartial.Prim(_fourPeekGraph);
            Assert.IsNotNull(result);
            Assert.AreEqual(3, result.Count);
            Assert.IsTrue(result.OrderBy(tuple => tuple.Item2)
                                .OrderBy(tuple => tuple.Item1)
                                .SequenceEqual(new[]
                                {
                                    Tuple.Create(0, 2, 2d),
                                    Tuple.Create(1, 2, 4d),
                                    Tuple.Create(3, 2, 1d)
                                }));
        }
    }
}
```