Algoritmica grafelor laboratorul 6
Tema: taietura de capacitate minima.

Enunt

Sa se conceapa un algoritm care primeste ca parametru de intrare un graf reprezetat prin lista succesorilor, o lista care contine liste formate din perechi (NxR^+) unde partea naturala reprezinta varful succesor iar partea real pozitiva capacitatea (valoarea) arcului. Algoritmul va returna un numar real egal cu taietura de capacitate minima in cazul in care graful dat este retea de transport, altfel minus infinit.

Dezvoltarea algoritmului

Pentru rezolvare se foloseste algoritmul lui Ford-Fulkerson (graful ecarturilor). Pentru aceasta trebuie determinat daca graful primit ca parametru este flux de transport. Folosind o functie pentru determinarea multimii varfurilor fara predecesor si o functie care determina multimea varfurilor fara succesor se poate determina varful sursa (varful fara predecesori) si varful destinatie (varful fara succesori) ai retelei de transport daca aceasta exista.

Pentru a determina daca un graf este conext sau nu se defineste o functie care verifica daca un graf dat si reprezentat prin lista succesorilor este conex. Functia intoarce o valoare booleana corespunzatoare (adevarat daca graful e conex, fals altfel).

Pentru determinarea fluxului maxim este nevoie de determinarea unui drum (oarecare). Pentru aceasta se va defini o functie care intoarce un drum, reprezentat printr-o secventa de varfuri, avand varful sursa, varful destinatie si graful reprezentat prin lista succesorilor. In cazul in care nu exista un drum intre cele doua varfuri se returneaza secventa vida.

Algoritmul principal foloseste toate functiile definite pentru a determina fluxul maxim care este egal cu taietura de capacitate minima (Teorema 4.4.1 pag. 159 din cartea atasata cursului).

Descrierea algoritmului

```
functie DeterminaVarfuriFaraPredecesori(graf)
    multime ← CreazaMultime()
    pentru i ← 0, graf.Length executa
        multime.Adauga(i)
    sfarsit pentru
    pentru i ← 0, graf.Length executa
        multime ← multime \ graf[i].Varfuri
    executa
    DeterminaVarfuriFaraPredecesori ← multime
sfarsit functie

functie DeterminaVarfuriFaraSuccesori(graf)
    multime ← CreazaMultime()
    pentru i ← 0, graf.Length executa
        daca graf[i].Length = 0 atunci
            multime.Adauga(i)
        sfarsit daca
    sfarsit pentru
    DeterminaVarfuriFaraSuccesori ← multime
sfarsit functie
```

```
functie EConex(graf)
    daca graf.Length = 0 atunci
        EConex ← adevarat
    altfel
        multime ← graf[0].Varfuri
        pentru i ← 1, graf.Length executa
            multime ← multime ∪ graf[i].Varfuri
        sfarsit pentru
        daca exista j ∈ multime astfel incat j >= graf.Length atunci
            EConex ← fals
        altfel
            EConex ← adevarat
        sfarsit daca
    sfarsit daca
sfarsit functie

functie DeterminaTaieturaDeCapacitateMinima(graf)
    varfuriFaraSuccesori ← DeterminaVarfuriFaraSuccesori(graf)
    varfuriFaraPredecesori ← DeterminaVarfuriFaraPredecesori(graf)
    daca varfuriFaraSuccesori.Length ≠ 1 sau varfuriFaraPredecesori ≠ 1
         sau not EConex(graf) atunci
        DeterminaTaieturaDeCapacitateMinima ← -inf
        stop
    sfarsit daca
    sursa ← varfuriFaraPredecesori.Primul
    destinatie ← varfuriFaraSucesori.Primul
    flux ← 0
    drum ← DeterminaDrum(graf, sursa, destinatie)
    repeta
        min ← +inf
        pentru fiecare varf din drum \ { drum.Ultimul } executa
            capacitateActuala ← graf.Gaseste(varf)
                                     .Gaseste(varf.Urmator)
                                     .Capacitate
            daca min > capacitateActuala atunci
                min ← capacitateActuala
            sfarsit daca
        sfarsit pentru
        pentru fiecare varf din drum \ { drum.Ultimul } executa
            capacitateActuala ← graf.Gaseste(varf).Gaseste(varf.Urmator)
            ramas ← (capacitateActuala - min)
            daca ramas = 0 atunci
                graf.Gaseste(varf).Elimina(varf.Urmator)
            altfel
                graf.Gaseste(varf).Gaseste(varf.Urmator).Capacitate ← ramas
            sfarsit daca
        sfarsit pentru
        flux ← flux + min
        drum ← DeterminaDrum(graf, sursa, destinatie)
    panacand drum.Length = 0
    DeterminaTaieturaDeCapacitateMinima ← flux
sfarsit functie

functie DeterminaDrum(graf, sursa, destinatie)
    drum ← CreazaLista()
    drum.Adauga(sursa)
    ultimul ← sursa
    cattimp exista varf ∈ graf[ultimul] nevizitat
            sau ultimul = destinatie executa
```

```
        drum.Adauga(varf ∈ graf[ultimul] nevizitat
        ultimul ← drum.Ultimul
    sfarsit cattimp
    daca ultimul = destinatie atunci
        DeterminaDrum ← drum
    altfel
        DeterminaDrum ← CreazaLista()
    sfarsit daca
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.Laborator6.Taietura
{
    public static class Taietura
    {
        public static double CapacitateMinima(IReadOnlyList<IReadOnlyCollection<Tuple<int,
double>>> graf)
        {
            if (graf == null)
                throw new ArgumentNullException("graf");
            IList<IList<Tuple<int, double>>> grafulEcarturilor = new List<IList<Tuple<int,
double>>>(graf.Select(listaSuccesori => (IList<Tuple<int,
double>>)listaSuccesori.ToList()));
            IReadOnlyCollection<int> varfuriFaraSuccesori =
_DeterminaVarfuriFaraSuccesori(grafulEcarturilor);
            IReadOnlyCollection<int> varfuriFaraPredecesori =
_DeterminaVarfuriFaraPredecesori(grafulEcarturilor);
            if (varfuriFaraSuccesori.Count != 1 || varfuriFaraPredecesori.Count != 1 || !
_EConex(grafulEcarturilor))
                return double.NegativeInfinity;
            int sursa = varfuriFaraPredecesori.First();
            int destinatie = varfuriFaraSuccesori.First();
            double flux = 0;
            IReadOnlyList<Tuple<int, double>> drum = _DeterminaDrum(grafulEcarturilor,
sursa, destinatie);
            do
            {
                double min = drum.Min(varf => varf.Item2);
                for (int indexVarf = 0, varfPrecedent = sursa; indexVarf < drum.Count;
indexVarf++)
                {
                    Tuple<int, double> varf = grafulEcarturilor[varfPrecedent].First(v => (v
== drum[indexVarf]));
                    double capacitateActuala = (varf.Item2 - min);
                    grafulEcarturilor[varfPrecedent].Remove(varf);
                    if (capacitateActuala > 0)
                        grafulEcarturilor[varfPrecedent].Add(Tuple.Create(varf.Item1,
capacitateActuala));
                    varfPrecedent = drum[indexVarf].Item1;
                }
                flux += min;
```

```
                drum = _DeterminaDrum(grafulEcarturilor, sursa, destinatie);
            } while (drum.Count > 0);
            return flux;
        }
        private static IReadOnlyCollection<int>
_DeterminaVarfuriFaraPredecesori(IList<IList<Tuple<int, double>>> graf)
        {
            ISet<int> multime = new HashSet<int>();
            for (int i = 0; i < graf.Count; i++)
                multime.Add(i);
            for (int i = 0; i < graf.Count; i++)
                multime = new HashSet<int>(multime.Except(graf[i].Select(succesor =>
succesor.Item1)));
            return multime.ToList();
        }
        private static IReadOnlyCollection<int>
_DeterminaVarfuriFaraSuccesori(IList<IList<Tuple<int, double>>> graf)
        {
            ISet<int> multime = new HashSet<int>();
            for (int i = 0; i < graf.Count; i++)
                if (graf[i].Count == 0)
                    multime.Add(i);
            return multime.ToList();
        }
        private static bool _EConex(IList<IList<Tuple<int, double>>> graf)
        {
            if (graf.Count == 0)
                return true;
            ISet<int> multime = new HashSet<int>(graf[0].Select(succesor =>
succesor.Item1));
            for (int i = 1; i < graf.Count; i++)
                multime = new HashSet<int>(multime.Union(graf[1].Select(succesor =>
succesor.Item1)));
            return (multime.Count(varf => (varf >= graf.Count)) == 0);
        }
        private static IReadOnlyList<Tuple<int, double>>
_DeterminaDrum(IList<IList<Tuple<int, double>>> graf, int sursa, int destinatie)
        {
            List<Tuple<int, double>> drum = new List<Tuple<int, double>>();
            int candidat = sursa;
            while (candidat != destinatie &&
                   graf[candidat].Except(drum).FirstOrDefault() != null)
            {
                Tuple<int, double> ultimulVarf = graf[candidat].Except(drum).First();
                drum.Add(ultimulVarf);
                candidat = ultimulVarf.Item1;
            }
            if (drum.Count > 0 && drum.Last().Item1 != destinatie)
                drum.Clear();
            return drum;
        }
    }
}
```

Date de test

```
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace AlgoritmicaGrafelor.Laborator6.Taietura.Tests
{
    [TestClass]
    public class CapacitateMinima
    {
        IReadOnlyList<IReadOnlyCollection<Tuple<int, double>>> _grafCuDouaVarfuri = new[]
        {
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(1, 1d) }),
            (IReadOnlyCollection<Tuple<int, double>>)new Tuple<int, double>[0]
        };
        IReadOnlyList<IReadOnlyCollection<Tuple<int, double>>>
_grafCuPatruVarfuriSiDouaDrumuriDeLaSursaLadestinatie = new[]
        {
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(1, 1d),
Tuple.Create(2, 2d) }),
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(3, 3d) }),
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(3, 3d) }),
            (IReadOnlyCollection<Tuple<int, double>>)new Tuple<int, double>[0]
        };
        IReadOnlyList<IReadOnlyCollection<Tuple<int, double>>> _grafCuCinciVarfuriSiCircuit
= new[]
        {
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(1, 1d) }),
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(2, 2d) }),
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(3, 3d) }),
            (IReadOnlyCollection<Tuple<int, double>>)(new []{ Tuple.Create(1, 1d),
Tuple.Create(4, 4d) }),
            (IReadOnlyCollection<Tuple<int, double>>)new Tuple<int, double>[0]
        };
        [TestMethod]
        public void TestGrafCuDouaVarfuri()
        {
            Assert.AreEqual(1d, Taietura.CapacitateMinima(_grafCuDouaVarfuri));
        }
        [TestMethod]
        public void TestGrafCuPatruVarfuriSiDouaDrumuriDeLaSursaLadestinatie()
        {
            Assert.AreEqual(3d,
Taietura.CapacitateMinima(_grafCuPatruVarfuriSiDouaDrumuriDeLaSursaLadestinatie));
        }
        [TestMethod]
        public void TestGrafCuCinciVarfuriSiCircuit()
        {
            Assert.AreEqual(1d, Taietura.CapacitateMinima(_grafCuCinciVarfuriSiCircuit));
        }
    }
}
```