Algoritmica grafelor laboratorul 1

Tema: Transformari intre reprezentari (matrice de adiacenta si lista succesorilor).

Enunt

a) Sa se conceapa un algoritm care primeste ca parametru de intrare o matrice patratica ce memoreaza valori boolene reprezentand matricea de adiacenta a unui graf si returneaza o lista care contine pentru fiecare linie din matricea de adiacenta o lista de valori reprezentand sucesorii acelui varf reprezentat prin linia respectiva.

b) 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 valori reprezentand lista Successorilor acelui varf si returneaza o matrice patratica ce memoreaza valori boolene reprezentand matricea de adiacenta a grafului primit. Matricea rezultata are tot atatea linii cate elemente se afla in lista primita.

Dezvoltarea algoritmului

Fiecare dintre cei doi algoritmi nu pot fi impartiti in subprobleme deoarece fiecare reprezenta un algoritm de transformare a unui graf dintr-o reprezentare in alta.

Descrierea algoritmului

a)

```
functie TransformaInListaSuccesorilor(matriceDeAdiacenta)
    numarVarfuri ← sqrt(matriceDeAdiacenta.NumarElemente)
    listaSuccesorilor ← CreazaLista(numarVarfuri)

    pentru indexLinie ← 0, numarVarfuri
        listaSuccesorilor[indexLinie] ← CreazaLista()
        pentru indexColoana ← 0, numarVarfuri
            daca matriceDeAdiacenta[indexLinie][indexColoana] atunci
                listaSuccesorilor[indexLinie].Adauga(indexColoana)
            sfarsit daca
        sfarsit pentru
    sfarsit pentru

    TransformaInListaSuccesorilor ← listaSuccesorilor
sfarsit functie
```

b)

```
functie TransformaInMatriceDeAdiacenta(listaSuccesorilor)
    matriceDeAdiacenta ← CreazaMatrice(listaSuccesorilor.NumarElemente,
                                       listaSuccesorilor.NumarElemente)
    pentru indexLinie ← 0, listaSuccesorilor.NumarElemente
        pentru indexColoana ← 0, listaSuccesorilor.NumarElemente
            matriceDeAdiacenta[indexLinie][indexColoana] ← fals
        sfarsit pentru

        pentru indexColoana ← 0, listaSuccesorilor[indexLinie].NumarElemente
            matriceDeAdiacenta[indexLinie][indexColoana] ← adevarat
        sfarsit pentru
    sfarsit pentru

    TransformaInMatriceDeAdiacenta ← matriceDeAdiacenta
sfarsit functie
```

Demostrarea corectitudinii

Pentru fiecare extremitate initiala (linie) exista in matricea de adicenta valoarea 1 pe coloana extremitatii terminale (coloana) iar in acelasi timp extremitatea terminala apare ca succesor al extremitatii initiale in lista sucesorilor celui din urma. Cu alte cuvinte totii indicii coloanelor unde se afla valoare 1 pentru fiecare linie din matricea de adiacenta semnifica succesorii acelei linii (varf).

Cod sursa (C#)

a)

```
static public IList<IList<int>> ToSuccessorsList(bool[,] adjacencyMatrix)
{
    if (adjacencyMatrix != null)
    {
        int numberOfPeaks = (int)Math.Sqrt(adjacencyMatrix.Length);

        if (Math.Pow(numberOfPeaks, 2) == adjacencyMatrix.Length)
        {
            IList<IList<int>> successorsList = new List<IList<int>>(numberOfPeaks);

            for (int lineIndex = 0; lineIndex < numberOfPeaks; lineIndex++)
            {
                IList<int> successorPeaks = new List<int>();

                for (int columnIndex = 0; columnIndex < numberOfPeaks; columnIndex++)
                    if (adjacencyMatrix[lineIndex, columnIndex])
                        successorPeaks.Add(columnIndex);

                successorsList.Add(successorPeaks);
            }

            return successorsList;
        }
        else
            throw new ArgumentException("The adjacencyMatrix is not a square matrix!");
    }
    else
        throw new ArgumentNullException("adjacencyMatrix");
}
```

b)

```
static public bool[,] ToAdjacencyMatrix(IList<IList<int>> successorsList)
{
    if (successorsList != null)
    {
        bool[,] adjacencyMatrix = new bool[successorsList.Count, successorsList.Count];

        for (int i = 0; i < successorsList.Count; i++)
            if (successorsList[i] != null)
                foreach (int successor in successorsList[i])
                    if (0 <= successor && successor <= successorsList.Count)
                        adjacencyMatrix[i, successor] = true;
                    else
                        throw new ArgumentException("The successor "
                                                    + successor
                                                    + " is not a peak in the graph!");
        return adjacencyMatrix;
    }
    else
        throw new ArgumentNullException("successorList");
}
```

Date de test

```
[TestClass]
public class TransformsTests
{
    [TestClass]
    public class ToSuccessorListTests
    {
        [TestMethod]
        public void TestWhenAdjacencyMatrixIsEmpty()
        {
            var SuccessorList = Transforms.ToSuccessorsList(new bool[,] { });

            Assert.AreEqual(0, SuccessorList.Count);
        }

        [TestMethod]
        public void TestWhenAdjacencyMatrixContainsOnlyFalse()
        {
            var SuccessorList = Transforms.ToSuccessorsList(new bool[,] {{false, false},
                                                                         {false, false}});

            Assert.AreEqual(2, SuccessorList.Count);
            Assert.AreEqual(0, SuccessorList[0].Count);
            Assert.AreEqual(0, SuccessorList[1].Count);
        }

        [TestMethod]
        public void TestWhenAdjacencyMatrixContainsOneTrueOnFirstPosition()
        {
            var SuccessorList = Transforms.ToSuccessorsList(new bool[,] {{true, false},
                                                                         {false, false}});

            Assert.AreEqual(2, SuccessorList.Count);
            Assert.AreEqual(1, SuccessorList[0].Count);
```

```
        Assert.AreEqual(0, SuccessorList[1].Count);
        Assert.AreEqual(0, SuccessorList[0][0]);
    }

    [TestMethod]
    public void TestWhenAdjacencyMatrixContainsContainsOnlyTrue()
    {
        var SuccessorList = Transforms.ToSuccessorsList(new bool[,] {{true, true},
                                                                     {true, true}});

        Assert.AreEqual(2, SuccessorList.Count);
        Assert.AreEqual(2, SuccessorList[0].Count);
        Assert.AreEqual(0, SuccessorList[0][0]);
        Assert.AreEqual(1, SuccessorList[0][1]);
        Assert.AreEqual(2, SuccessorList[1].Count);
        Assert.AreEqual(0, SuccessorList[1][0]);
        Assert.AreEqual(1, SuccessorList[1][1]);
    }

    [TestMethod]
    public void TestWhenAdjacencyMatrixHasCircularReference()
    {
        var SuccessorList = Transforms.ToSuccessorsList(new bool[,] {{false, true},
                                                                     {true, false}});

        Assert.AreEqual(2, SuccessorList.Count);
        Assert.AreEqual(1, SuccessorList[0].Count);
        Assert.AreEqual(1, SuccessorList[0][0]);
        Assert.AreEqual(1, SuccessorList[1].Count);
        Assert.AreEqual(0, SuccessorList[1][0]);
    }

    [TestMethod, ExpectedException(typeof(ArgumentNullException))]
    public void TestWhenAdjacencyMatrixIsNull()
    {
        Transforms.ToSuccessorsList(null);
    }

    [TestMethod, ExpectedException(typeof(ArgumentException))]
    public void TestWhenAdjacencyMatrixIsNotSquareMatrix()
    {
        Transforms.ToSuccessorsList(new bool[,] { { false }, { true } });
    }
}

[TestClass]
public class ToAdjacencyMatrixTests
{
    [TestMethod]
    public void TestWhenSuccessorsListIsEmpty()
    {
        var adjacencyMatrix = Transforms.ToAdjacencyMatrix(new List<IList<int>>());

        Assert.AreEqual(0, adjacencyMatrix.Length);
    }
```

```
        [TestMethod]
        public void TestWhenSuccessorsListHas4Nulls()
        {
            var adjacencyMatrix = Transforms.ToAdjacencyMatrix(new List<IList<int>>()
                                                               {
                                                                    null,
                                                                    null,
                                                                    null,
                                                                    null
                                                               });

            Assert.AreEqual(16, adjacencyMatrix.Length);
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    Assert.IsFalse(adjacencyMatrix[i, j]);
        }
    }
}
```