Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №6

Задача о назначениях

Выполнил:

студент гр. 953504

Кондрашов И.Д.

Проверил:

кандидат физико-математических наук

Дугинов О.И.

Минск 2022

1. **Пример работы программы**

Входные данные:

Text

Description automatically generated

Ответ:

Text

Description automatically generated

1. **Исходный код программы (C#)**

Реализованные методы:

1. OneIteration(Vector<double> alphas, Vector<double> betas, int matrixLength, Matrix<double> matrix) – метод, реализующий одну итерацию алгоритма из лабораторной (рекурсивная функция)
2. Assignment(Matrix<double> matrixC) – метод, выполняющий задание лабораторной(решение задачи о назначениях)

Методы из лабораторной работы №5

1. DFS(ref double[][] graph, int graphRowIndex, ref bool[] visited, ref int[] matching) – метод, реализующий поиск в глубину
2. MaximumMatching(Matrix<double> graph)– метод, выполняющий задание лабораторной(поиск максимального паросочетания)

using MathNet.Numerics.LinearAlgebra;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lr6

{

class Program

{

static void Main(string[] args)

{

var matrixC = Matrix<double>.Build.DenseOfArray(new double[,] { { 7, 2, 1, 9, 4 },

{ 9, 6, 9, 5, 5 },

{ 3, 8, 3, 1, 8 },

{ 7, 9, 4, 2, 2 },

{ 8, 4, 7, 4, 8 } });

var match = Assignment(matrixC);

Console.Write("Назначения:");

foreach (var item in match)

{

Console.Write($"[{String.Join(", ", item)}] ");

}

Console.WriteLine();

}

private static List<int[]> Assignment(Matrix<double> matrixC)

{

int matrixLength = matrixC.RowCount;

var alphaArray = Vector<double>.Build.Dense(matrixLength, 0);

var betaArray = Vector<double>.Build.DenseOfEnumerable(matrixC.ToRowArrays().Select((\_, i) => matrixC.Column(i).Min()));

var res = OneIteration(alphaArray, betaArray, matrixLength, matrixC);

return res;

}

private static List<int[]> OneIteration(Vector<double> alphas, Vector<double> betas, int matrixLength, Matrix<double> matrix)

{

var graph = Matrix<double>.Build.Dense(matrixLength, matrixLength);

for (int i = 0; i < matrixLength; i++)

{

for (int j = 0; j < matrixLength; j++)

{

if (alphas[i] + betas[j] == matrix[i, j])

{

graph[i, j] = 1;

}

}

}

var match = MaximumMatching(graph);

if (match.Count == matrixLength)

{

return match;

}

var orientedGraph = Matrix<double>.Build.DenseOfMatrix(graph);

for (int i = 0; i < matrixLength; i++)

{

for (int j = 0; j < matrixLength; j++)

{

var apex = match.Find(x => x[0] == i && x[1] == j);

if (apex is not default(int[]))

{

orientedGraph[i, j] \*= -1;

}

}

}

var starred = new List<List<int?>>();

var rows = orientedGraph.ToRowArrays().ToList();

foreach (var row in rows)

{

if (row.Contains(-1))

{

int index = rows.IndexOf(row);

starred.Insert(index, null);

}

else

{

var star = new List<int?>();

for (int i = 0; i < row.Length; i++)

{

if (row[i] == 1)

{

star.Insert(i, i);

}

else

{

star.Insert(i, null);

}

}

star = star.Where(x => x is not null).ToList();

starred.Add(star);

}

}

var upperPortion = new List<int>();

foreach (var star in starred)

{

if (star is not null)

{

int index = starred.IndexOf(star);

upperPortion.Add(index);

}

}

var bottomPortion = new List<int?>();

foreach (var stars in starred)

{

if (stars is not null)

{

foreach (var star in stars)

{

bool isRes = Convert.ToBoolean(star);

if (isRes)

{

bottomPortion.Add(star);

}

}

}

}

var tempAlphaArray = new int[matrixLength];

for (int i = 0; i < tempAlphaArray.Length; i++)

{

if (upperPortion.Contains(i))

{

tempAlphaArray[i] = 1;

}

else

{

tempAlphaArray[i] = -1;

}

}

var tempBetaArray = new int[matrixLength];

for (int i = 0; i < tempBetaArray.Length; i++)

{

if (bottomPortion.Contains(i))

{

tempBetaArray[i] = -1;

}

else

{

tempBetaArray[i] = 1;

}

}

var tettaList = new List<double>();

for (int i = 0; i < matrixLength; i++)

{

for (int j = 0; j < matrixLength; j++)

{

if (upperPortion.Contains(i) && !bottomPortion.Contains(j))

{

tettaList.Add((matrix[i, j] - alphas[i] - betas[j]) / 2);

}

}

}

double tetta = tettaList.Min();

var newAlphaArray = Vector<double>.Build.DenseOfEnumerable(alphas.Select((x, i) => x + tetta \* tempAlphaArray[i]));

var newBetaArray = Vector<double>.Build.DenseOfEnumerable(betas.Select((x, i) => x + tetta \* tempBetaArray[i]));

return OneIteration(newAlphaArray, newBetaArray, matrixLength, matrix);

}

/\* Методы из ЛР №5 'Поиск максимального паросочетания' \*/

private static List<int[]> MaximumMatching(Matrix<double> graph)

{

int[] matching = new int[graph.RowCount];

Array.Fill(matching, -1);

bool[] visited = new bool[graph.RowCount];

var graphArray = graph.ToRowArrays();

for (int i = 0; i < graph.RowCount; i++)

{

Array.Fill(visited, false);

DFS(ref graphArray, i, ref visited, ref matching);

}

var match = matching.Select((i, j) => new int[] { i, j }).Where(v => !v.Contains(-1));

var matchArray = match.ToList();

return matchArray;

}

private static bool DFS(ref double[][] graph, int graphRowIndex, ref bool[] visited, ref int[] matching)

{

for (int i = 0; i < graph[graphRowIndex].Length; i++)

{

if (graph[graphRowIndex][i] == 1 && !visited[i])

{

visited[i] = true;

if (matching[i] == -1 || DFS(ref graph, matching[i], ref visited, ref matching))

{

matching[i] = graphRowIndex;

return true;

}

}

}

return false;

}

}

}}