МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

Институт информационных технологий и управления в технических системах

кафедра Информационные системы

09.03.02 Информационные системы и технологии (уровень бакалавриата)

Лабораторная работа №3

По дисциплине: «Теория принятия решений»

ИССЛЕДОВАНИЕ ПРИМЕНЕНИЯ ТЕОРИИ ВАЖНОСТИ КРИТЕРИЕВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ВЫБОРА АЛЬТЕРНАТИВ

Вариант – 2

Выполнил

студент 3 курса группы ИС/б-33-о

Генералов Николай Николаевич

|  |  |
| --- | --- |
|  |  |

Проверил

доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кротов К. В.

(должность) (подпись) (фамилия, инициалы)

г. Севастополь

2018 г.

# ЦЕЛЬ РАБОТЫ

Исследовать применение аппарата теории важности критериев при принятии решений по выбору альтернатив.

# ЗАДАНИЕ

Определить множество несравнимых решений XΘ, используя количественную информацию о важности критериев Θ в следующем виде:

Θ = {K3 3 K4, K1­2 K4, K4 2K2, K2 ~ K5}.

Таблица 1. Скалярные оценки критериев  для решений  ()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Варианты | Критерии | | | | |
|  |  |  |  |  |
|  | 3 | 5 | 5 | 4 | 4 |
|  | 4 | 4 | 4 | 5 | 4 |
|  | 5 | 4 | 3 | 3 | 5 |
|  | 3 | 5 | 3 | 5 | 3 |
|  | 4 | 2 | 4 | 5 | 5 |
|  | 3 | 5 | 3 | 5 | 3 |
|  | 5 | 3 | 4 | 3 | 4 |
|  | 4 | 5 | 3 | 4 | 3 |

# ПОСТАНОВКА ЗАДАЧИ

* + - * На основе информации  о количественной важности критериев сформировать *N*-модель в виде вектора, каждый *i-*ый элемент которого соответствует *i-*му критерию и определяет число повторений исходных скалярных оценок  в формируемом векторе  (при );
      * разработать процедуру определения доминируемых решений, выполняющую для каждого решения  сравнение его значений скалярных оценок  вектора  с такими же скалярными оценками  решений ; тем самым должны быть определены решения , доминируемые текущим рассматриваемым решением  (при  и ); результатом выполнения процедуры является множество  не сравнимых между собой с использованием отношения предпочтения решений;
* разработать процедуру, использующую информацию  о важности критериев, входными данными для которой будет являться сформированный вектор значений, интерпретируемый как *N-*модель; разрабатываемая процедура реализует формирование векторов  (), представляющих собой модификацию исходных векторных оценок  () по соответствующему виду *N-*модели; таким образом, результатом реализации процедуры являются модифицированные с учетом информации  о количественной важности критериев векторные оценки  ();
* разработать процедуру, упорядочивающую по убыванию скалярные оценки  () для каждой сформированной векторной оценки  ();
* для модифицированных векторных оценок  каждого решения  () проконтролировать выполнение условия доминирования им других решений  для их векторных оценок  (при  и ) (т.е. выполняется поэлементное сравнение оценок  и  из соответствующих векторов  и ); при выполнении условия , процедура реализует исключение решения  из множества : ;
* результатом выполнения разрабатываемой программы является определение множества не сравнимых решений , сформированного на основе информации  о количественной важности критериев;
* выполнить вывод множества , полученного в результате исключения из него доминируемых решений  при учете дополнительной информации  о количественной важности критериев.

# ХОД РАБОТЫ

* 1. Аналитические расчеты.

Составим матрицу количественной информации­ – A1, о важности критериев на основании Θ:

Θ = {K3 3 K4, K1­2 K4, K4 2K2, K2 ~ K5} (1)

Рисунок 1 – Матрица количественной информации­ о важности критериев

Используя матрицу А1 построим N-кратную модель:

(2)

Путем повторения k*j-*скалярных оценок Nj раз сформируем расширенную модель с учетом количественной важности критериев:

KΘ (x1) = {3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4}

KΘ (x2) = {4, 4 ,4, 4, 5, 4, 4 ,4, 4, 4, 4, 5, 5, 4}

KΘ (x3) = {5, 5, 5, 5, 4, 3, 3, 3, 3, 3, 3, 3, 3, 5}

KΘ (x4) = {3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 5, 5, 3}

KΘ (x5) = {4, 4 ,4, 4, 2, 4, 4 ,4, 4, 4, 4, 5, 5, 5}

KΘ (x6) = {3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 5, 5, 3}

KΘ (x7) = {5, 5, 5, 5, 3, 4, 4 ,4, 4, 4, 4, 3, 3, 4}

KΘ (x8) = {4, 4 ,4, 4, 5, 3, 3, 3, 3, 3, 3, 4, 4, 3}

Упорядочим полученные оценки:

KΘ (x1) = {5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 3, 3, 3, 3}

KΘ (x2) = {5, 5, 5, 4, 4 ,4, 4, 4, 4 ,4, 4, 4, 4, 4}

KΘ (x3) = {5, 5, 5, 5, 5, 4, 3, 3, 3, 3, 3, 3, 3, 3}

KΘ (x4) = {5, 5, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3}

KΘ (x5) = {5, 5, 5, 4, 4 ,4, 4, 4, 4 ,4, 4, 4, 4, 2}

KΘ (x6) = {5, 5, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3}

KΘ (x7) = {5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3}

KΘ (x8) = {5, 4, 4 ,4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3}

Используя полученные данные сформируем множество недоминируемых решений XΘ:

XΘ = { x1, x2, x5, x7 }

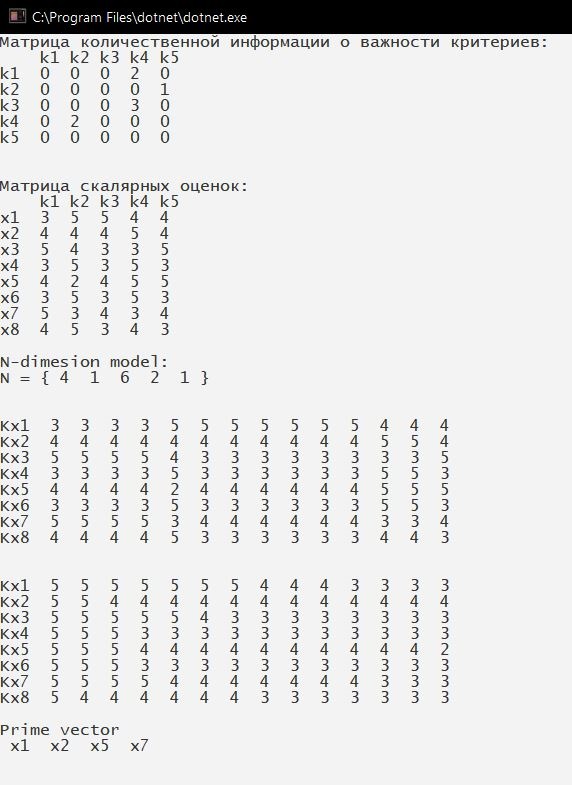
1. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Рисунок 1 – Вывод результатов выполнения программы в консоль

# ВЫВОД

В ходе выполнения лабораторной работы исследовалось применение аппарата теории важности критериев при принятии решений по выбору альтернатив. Исследование проводилось с использованием количественной информации о важности критериев Θ.

Так же в ходе лабораторной работы была написана программа, на языке C# осуществляющая, на основании загруженных из файла матрицы скалярных оценок и матрицы о количественной информации важности критериев. В результате получили множество недоминируемых решений имеющее вид:

XΘ = { x1, x2, x5, x7 }

Результаты выполнения программы и аналитических расчетов совпали, следовательно можно сделать вывод, что программа составлена правильно и поставленная цель достигнута.

Приложение А

Текст программных модулей

«Program.cs»

using System;

using System;

using System.Collections.Generic;

namespace TPR\_lab3

{

class Program

{

static void Main(string[] args)

{

MatrixLoader LoadedOrderInformationMatrix = new MatrixLoader("A1.txt");

MatrixLoader LoadedMarksMatrix = new MatrixLoader("A2.txt");

MyMatrix<int> InformationMatrix = new MyMatrix<int>(LoadedOrderInformationMatrix.Load());

MyMatrix<int> MarksMatrix = new MyMatrix<int>(LoadedMarksMatrix.Load());

Console.WriteLine("Матрица количественной информации о важности критериев:");

MyMatrix<int>.PrintMatrix(InformationMatrix.Matrix, "k", "k");

Console.WriteLine(Environment.NewLine);

Console.WriteLine("Матрица скалярных оценок:");

MyMatrix<int>.PrintMatrix(MarksMatrix.Matrix, "x", "k");

Console.WriteLine("");

N\_dimesion\_model nDimModel = new N\_dimesion\_model(InformationMatrix);

Console.WriteLine("N-dimesion model:");

Console.Write("N = {");

N\_dimesion\_model.PrintVector(nDimModel.VectorOfMarksCount);

Console.WriteLine("}");

Console.WriteLine("");

MyMatrix<int> extendedModel = nDimModel.GetExtendedModel(MarksMatrix);

MyMatrix<int>.PrintMatrix(extendedModel.Matrix,"Kx");

Console.WriteLine("");

N\_dimesion\_model.SortMatrix(extendedModel);

MyMatrix<int>.PrintMatrix(extendedModel.Matrix, "Kx");

List<int> rootsVector = N\_dimesion\_model.FindRoots(extendedModel);

Console.WriteLine("");

Console.WriteLine("Prime vector");

N\_dimesion\_model.PrintRoots(rootsVector, "x");

Console.WriteLine("");

}

}

}

«MatrixLoader.cs»

using System;

using System.Collections.Generic;

using System.IO;

namespace TPR\_lab3

{

public class MatrixLoader

{

public string FileName { get; private set; }

protected StreamReader \_fread { get; private set; }

public MatrixLoader(string \_filename)

{

FileName = \_filename;

try

{

\_fread = new StreamReader(new FileStream(FileName, FileMode.Open));

}

catch (FileNotFoundException err)

{

Console.WriteLine(err);

Environment.Exit(0);

}

}

public List<List<int>> Load()

{

try

{

List<List<int>> matrix = new List<List<int>>();

int matrixRowIndex = 0;

while (!\_fread.EndOfStream)

{

string tempString = \_fread.ReadLine();

matrix.Add(new List<int>());

string[] NumericStrings = tempString.Split(" ");

foreach (var num in NumericStrings)

{

int resultNum = 0;

if (Int32.TryParse(num, out resultNum))

{

matrix[matrixRowIndex].Add(resultNum);

}

}

++matrixRowIndex;

}

return matrix;

}

catch (IOException err)

{

Console.WriteLine(err);

return null;

}

}

~MatrixLoader()

{

if (\_fread != null)

{

\_fread.Close();

}

}

}

}

«MyMatrix.cs»

using System;

using System.Collections.Generic;

namespace TPR\_lab3

{

public class MyMatrix<T>

{

public MyMatrix()

{

this.Matrix = new List<List<T>>();

}

public MyMatrix(T[,] initMatrix)

{

Matrix = ConvertToMatrix(initMatrix);

}

public MyMatrix(List<List<T>> initMatrix)

{

Matrix = initMatrix;

}

public MyMatrix(MyMatrix<T> initMatrix)

{

this.Matrix = new List<List<T>>();

for (int i = 0; i < initMatrix.Matrix.Count; ++i)

{

this.Matrix.Add(initMatrix.Matrix[i]);

}

}

public List<List<T>> Matrix { get; set; }

public static List<List<T>> ConvertToMatrix(T[,] matrix)

{

List<List<T>> tempMatrix = new List<List<T>>();

int rowCount = matrix.GetLength(0);

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

{

tempMatrix.Add(new List<T>());

for (int j = 0; j < (matrix.Length / rowCount); ++j)

{

tempMatrix[i].Add(matrix[i, j]);

}

}

return tempMatrix;

}

public static void PrintMatrix(List<List<T>> Matrix, string rowChar = " ", string colChar = " ")

{

Console.Write(" ");

for (int i = 0; i < Matrix[0].Count; ++i)

{

if (colChar != " ")

{

Console.Write($"{colChar}{i + 1} ");

}

}

Console.WriteLine();

for (int i = 0; i < Matrix.Count; ++i)

{

Console.Write($"{rowChar}{i + 1} ");

for (int j = 0; j < Matrix[i].Count; ++j)

{

Console.Write($"{Convert.ToInt16(Matrix[i][j]),2} ");

}

Console.WriteLine();

}

}

public static List<List<bool>> ConvertToBool(List<List<int>> IntMatrix)

{

List<List<bool>> resultMatrix = new List<List<bool>>();

for (int i = 0; i < IntMatrix.Count; ++i)

{

resultMatrix.Add(new List<bool>());

for (int j = 0; j < IntMatrix[i].Count; ++j)

{

resultMatrix[i].Add((IntMatrix[i][j] > 0) ? true : false);

}

}

return resultMatrix;

}

public static void FillSameValue(List<bool> vector, bool value)

{

for (int i = 0; i < vector.Count; ++i)

{

vector[i] = value;

}

}

}

}

«UsefulValues.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TPR\_lab3

{

public class N\_dimesion\_model

{

public List<int> VectorOfMarksCount { get; set; }

public N\_dimesion\_model(MyMatrix<int> InformationMatrix)

{

VectorOfMarksCount = CalcNdimensionModel(InformationMatrix);

}

private List<int> CalcNdimensionModel(MyMatrix<int> InformationMatrix)

{

List<int> tempVector = new List<int>(InformationMatrix.Matrix.Count);

for (int k = 0; k < tempVector.Capacity; ++k)

{

tempVector.Add(0);

}

//Search zero row

int? zeroRowNumber = null;

for(int i = 0; i < InformationMatrix.Matrix.Count; ++i)

{

bool isZeroRow = false;

for(int j = 0; j < InformationMatrix.Matrix.Count; ++j)

{

if (InformationMatrix.Matrix[i][j] != 0)

{

isZeroRow = false;

break;

}

isZeroRow = true;

}

if (isZeroRow)

{

calculateCoefficient(InformationMatrix, i, tempVector);

break;

}

}

return tempVector;

}

private void calculateCoefficient(MyMatrix<int> informationMatrix, int colNumber, List<int> tempVector,

int result = 1)

{

tempVector[colNumber] = result;

for (int i = 0; i < informationMatrix.Matrix.Count; ++i)

{

if (informationMatrix.Matrix[i][colNumber] != 0)

{

int tempResult = result \* informationMatrix.Matrix[i][colNumber];

calculateCoefficient(informationMatrix, i, tempVector, tempResult);

}

}

}

public MyMatrix<int> GetExtendedModel(MyMatrix<int> marksMatrix)

{

int extendedModelColSize = VectorOfMarksCount.Sum();

MyMatrix<int> newMatrix = new MyMatrix<int>();

for (int i = 0; i < marksMatrix.Matrix.Count; ++i)

{

newMatrix.Matrix.Add(new List<int>());

for (int j = 0; j < VectorOfMarksCount.Count; ++j)

{

for (int k = 0; k < VectorOfMarksCount[j]; ++k)

{

newMatrix.Matrix[i].Add(marksMatrix.Matrix[i][j]);

}

}

}

return newMatrix;

}

public static void SortMatrix(MyMatrix<int> extendMatrix)

{

for (int i = 0; i < extendMatrix.Matrix.Count; ++i)

{

extendMatrix.Matrix[i].Sort((x1, x2) => { return x2 - x1; });

}

}

public static List<int> FindRoots(MyMatrix<int> extendMatrix)

{

var resultVector = new List<int>();

for (int i = 0; i < extendMatrix.Matrix.Count; ++i)

{

resultVector.Add(1);

}

for (int i = 0; i < extendMatrix.Matrix.Count - 1; ++i)

{

if (resultVector[i] != 0)

{

for (int j = i + 1; j < extendMatrix.Matrix.Count; ++j)

{

if (resultVector[j] != 0)

{

cmpMarksVectors(extendMatrix, i, j, resultVector);

}

}

}

}

return resultVector;

}

private static void cmpMarksVectors(MyMatrix<int> extendMatrix, int vec1Index, int vec2Index, List<int> rootsVector)

{

int size = extendMatrix.Matrix[vec1Index].Count;

bool vec1IsRoot = true;

bool vec2IsRoot = true;

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

{

if (vec1IsRoot)

{

if (extendMatrix.Matrix[vec1Index][i] < extendMatrix.Matrix[vec2Index][i])

{

vec1IsRoot = false;

}

}

if (vec2IsRoot)

{

if (extendMatrix.Matrix[vec2Index][i] < extendMatrix.Matrix[vec1Index][i])

{

vec2IsRoot = false;

}

}

if (!vec1IsRoot && !vec2IsRoot)

{

break;

}

}

if (vec1IsRoot && !vec2IsRoot)

{

rootsVector[vec2Index] = 0;

} else if (!vec1IsRoot && vec2IsRoot)

{

rootsVector[vec1Index] = 0;

}

}

public static void PrintVector(List<int> rootVector, string fillStr = " ")

{

for (int i = 0; i < rootVector.Count; ++i)

{

Console.Write($"{fillStr}{rootVector[i]} ");

}

}

public static void PrintRoots(List<int> rootVector, string fillStr = " ")

{

for (int i = 0; i < rootVector.Count; ++i)

{

if (rootVector[i] != 0)

{

Console.Write($" {fillStr}{i + 1} ");

}

}

}

}

}