МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Факультет информационных технологий и робототехники (ФИТР)**

**Отчёт по лабораторной работе №3**

По дисциплине: «Методы и алгоритмы принятия решений»

На тему: «МОДЕЛИ ДИСКРЕТНОГО ПРОГРАММИРОВАНИЯ»

Вариант 10

**Выполнил:**  студент группы 10701118 Дубоделов А.В.

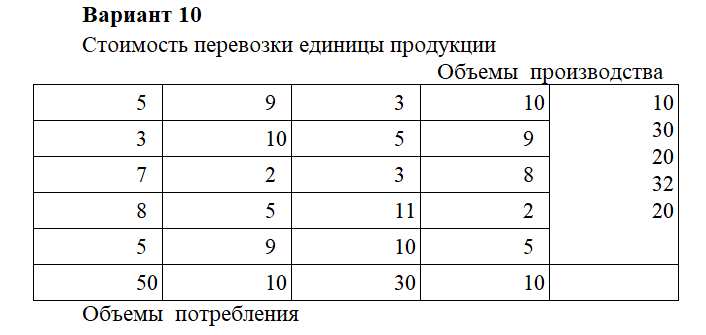
**Приняла ст. преподаватель:** Борисова И.М.

Минск 2020

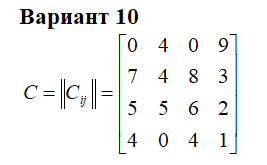
**Цель работы:** изучить способы решения зада дискретного программирования, изучить Венгерский метод и поиск решения в среде MS EXCEL.

**Постановка задачи:**

1.Транспортную задачу используя  Поиск решения в  MS EXCEL.



2.Задачу о назначениях используя  Поиск решения в  MS EXCEL и венгерский метод.



3.Разработать приложение для решения задачи о назначениях Венгерским методом

**Выполнение транспортной задачи**

1. Составляем таблицу исходных данных транспортной задачи *Рисунок 1.*

****

Рисунок 1 – Условия транспортной задачи

1. Проверяем задачу на условие закрытости, сумма объемов производства должна быть равна сумма объемов потребления *Рисунок 2*.

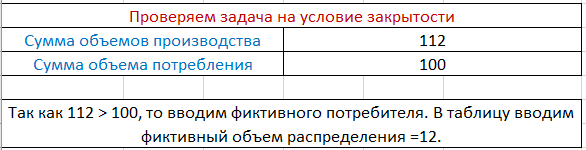


Рисунок 2 – Результат решения задачи

Фиктивный потребитель *Рисунок 3*.



Рисунок 3 – Добавление фиктивного объема распределения

Целевая функция:

|  |
| --- |
| F(x) = 5\*x11 + 9\*x12 + 3\*x13 + 10\*x14 + 0\*x15 +  3\*x21 + 10\*x22 + 5\*x23 + 9\*x24 + 0\*x25 +  7\*x31 + 2\*x32 + 3\*x33 + 8\*x34 + 0\*x35 +  8\*x41 + 5\*x42 + 11\*x43 + 2\*x44 + 0\*x45 +  5\*x51 + 9\*x52 + 10\*x53 + 5\*x54 + 0\*x15 -> min |

1. Результат поиска решения представлен на *Рисунок 4.*

****

Рисунок 4 – Результат выполнения *Поиска решения*

Значение целевой функции представлено на *Рисунке 5.*

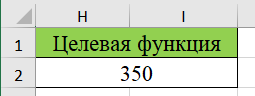
****

Рисунок 5 – Значение целевой функции

В результате выполнения задачи минимальная стоимость равна 350. План перевозок представлен на *Рисунке 4*.

1. На *Рисунке 6* представлены формулы ячеек в таблице Excel.

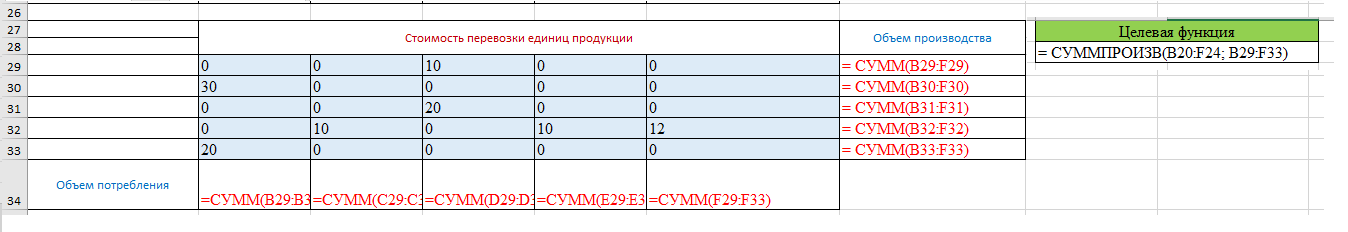
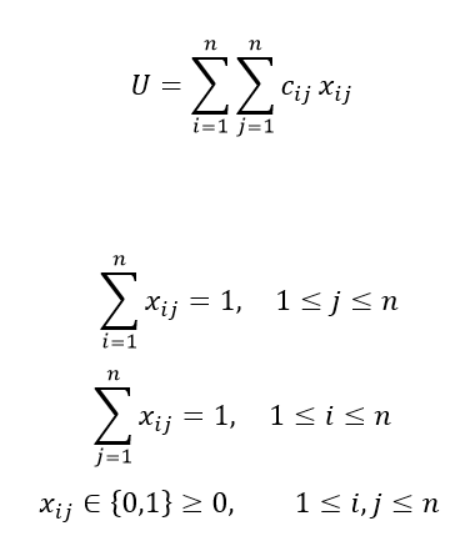
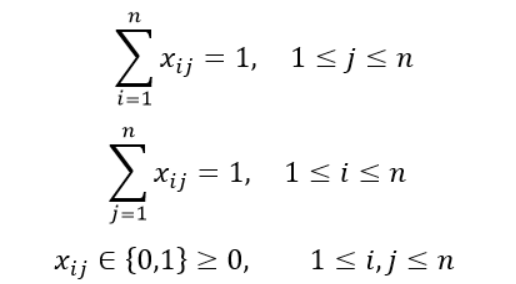


Рисунок 6 – Формулы в таблице Excel.

**Выполнение задачи на назначение (Венгерский метод)**

Целевая функция: Ограничения:

 ****

1. Составим таблицу исходных данных задачи о назначениях *Рисунок 7.*

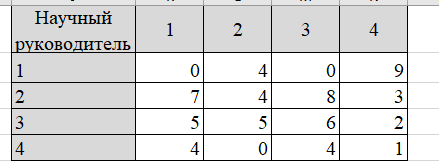


Рисунок 7 – Поиск решения для решения задачи 10 варианта

1. Запишем в столбце *минимального времени по строке Рисунок 8.*

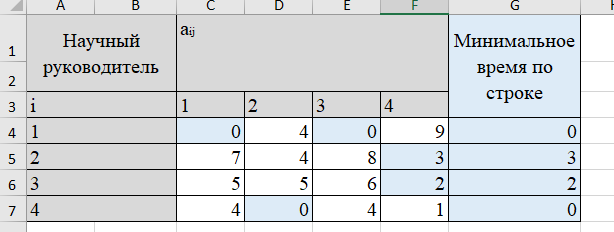
**

Рисунок 8 – Минимальные значение по строке

1. Отнимем минимальное значение строки от каждого элемента в ней, а также найдем минимальные значения по графам *Рисунок 9.*

**

Рисунок 9 – Минимальные значения по графам

1. Результаты выполнения назначений представлены на *Рисунке 10.*

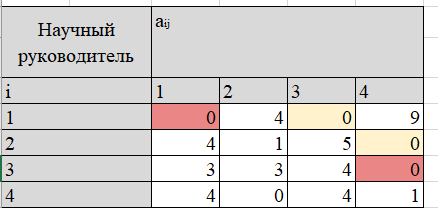
**

Рисунок 10 – Результат назначений

Так как в матрице только 2 независимых 0, из четырех необходимых, то решение недопустимое.

1. Найдем минимальный набор строк и столбцов, содержащий все нули. В незатронутых клетках найдем минимальное значение элемента. Результат данного этапа представлен на *Рисунке 11.*

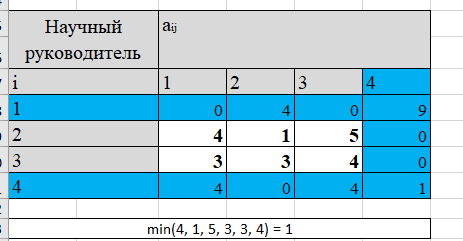
**

Рисунок 11 – Результат выделения минимального набора строк и столбцов и нахождение минимального элемента среди оставшихся

1. Результат выполнения преобразований представлен на *Рисунке 12.*

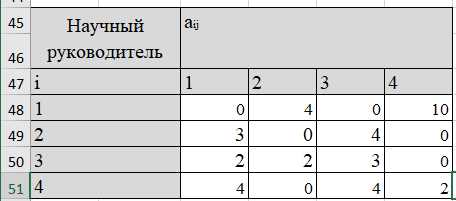
**

Рисунок 12 – Таблица после преобразований

1. После преобразований повторим пункт 8. Результаты выполненного назначения представлены на *Рисунке 13.*

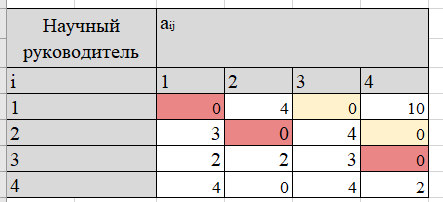
**

Рисунок 13 – Результаты повторного назначения

Преобразования представлены на *Рисунке 14,* а их результат на *Рисунке 15.*

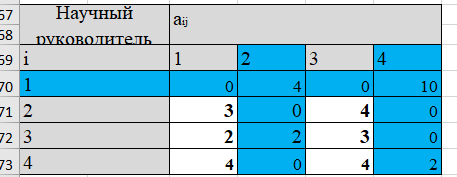
**

Рисунок 14 – Результат выделения минимального набора строк и столбцов и нахождение минимального элемента среди оставшихся

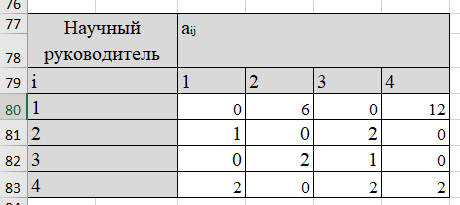


Рисунок 15 – Таблица после преобразований

1. Снова сделаем назначения *Рисунок 16.*

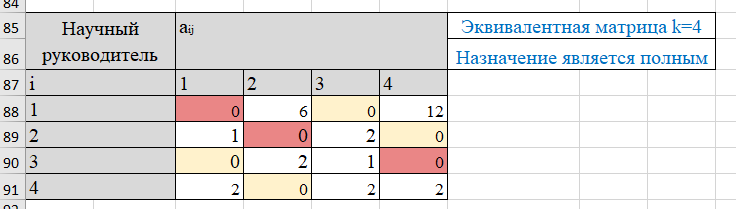
**

Рисунок 16 – Результат назначений

Назначение можно считать полным, так как количество свободных нулей равно 4.

1. Соотнесем позиции свободных 0-й, с позициями в исходной матрице и найдем минимальное кол-во времени, необходимого для реализации всех 4-х проектов. **T = 5x + 0x + 0x + 3x = 8** месяцев.

**Выполнение задачи на назначение (Поиск решения)**

1. Формула целевой функции *Рисунок 17*.

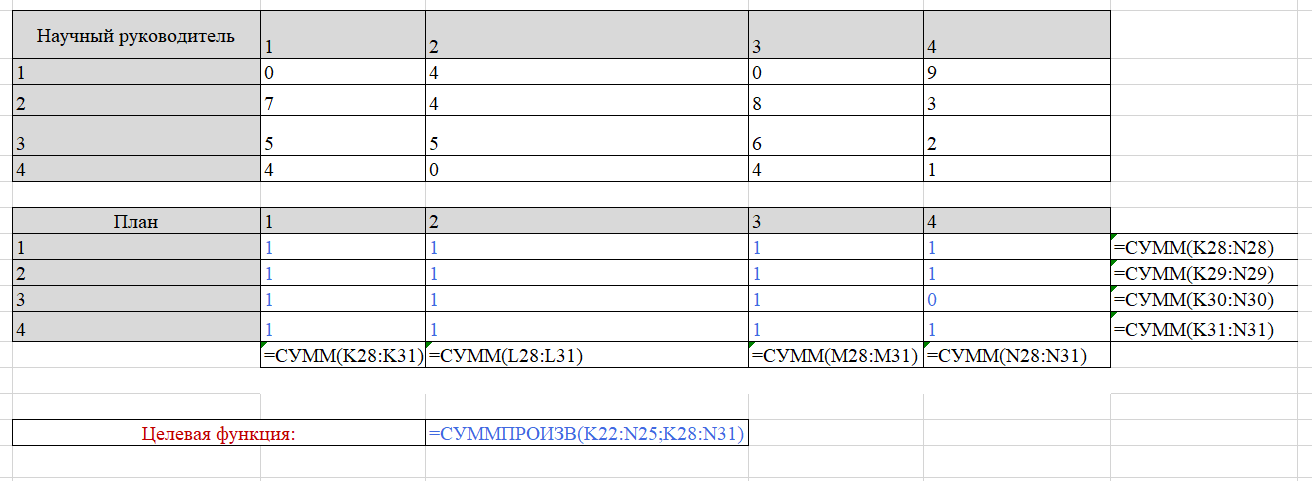


Рисунок 17 – Формулы целевой функции и ограничений

1. Суммарные временные затраты(целевая функция) представлены на *Рисунке 18.*

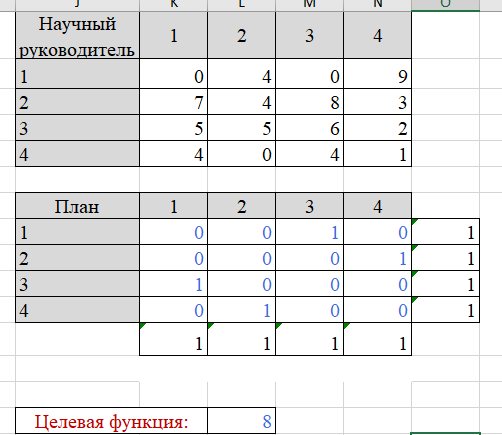
**

Рисунок 18 – Результат выполнения функции *Поиск решения.*

**Разработка программы для решения задач Венгерским методом**

1. В ходе лабораторной работы была разработана программа для решения задач на назначения, используя Венгерский метод.

Передаваемые данные: Матрица исходных данных в виде двухмерного массива.

Результат выполнения программы представлен на *Рисунке 19.*

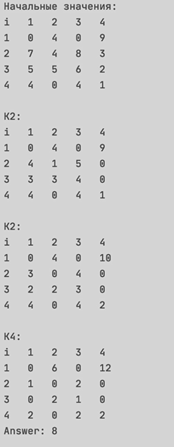
**

Рисунок 19 – Результат выполнения программы

**Вывод**

Познакомился с задачами дискретного программирования и способами их решения в частность Венгерский метод. С помощью инструмента MS Excel решил задачу о транспортных поставках, результатом которой является план поставок продукции от поставщиков к потребителям с минимальной стоимостью перевозки, при помощи венгерского метода было представлено решение задачи о назначениях, в результате решения которой, было выяснено, что реализовать все 4 проекта возможно за 8 месяцев. Также решение данной задачи реализовано в виде программы.

**Листинг**

Main.java

public class Test {

    public static void main(String[] args) {

        int answer = 0;

        int[][] dataMatrix = {

                {0,  4,   0,   9},

                {7,  4,   8,   3},

                {5,  5,   6,   2},

                {4,  0,   4,   1}

        };

        int[][] dataMatrix1 = {

                {0,  4,   0,   9},

                {7,  4,   8,   3},

                {5,  5,   6,   2},

                {4,  0,   4,   1}

        };

        HungarianAlgorithm ha = new HungarianAlgorithm(dataMatrix);

        int[][] assignment = ha.findOptimalAssignment();

HungarianAlgorithm.java

import java.util.Arrays;

import java.util.LinkedHashSet;

import java.util.Set;

public class HungarianAlgorithm {

    int[][] matrix;

    int[] squareInRow, squareInCol, rowIsCovered, colIsCovered, staredZeroesInRow;

    public HungarianAlgorithm(int[][] matrix) {

        if (matrix.length != matrix[0].length) {

            try {

                throw new IllegalAccessException("The matrix is not square!");

            } catch (IllegalAccessException ex) {

                System.err.println(ex);

                System.exit(1);

            }

        }

        this.matrix = matrix;

        squareInRow = new int[matrix.length];

        squareInCol = new int[matrix[0].length];

        rowIsCovered = new int[matrix.length];

        colIsCovered = new int[matrix[0].length];

        staredZeroesInRow = new int[matrix.length];

        Arrays.fill(staredZeroesInRow, -1);

        Arrays.fill(squareInRow, -1);

        Arrays.fill(squareInCol, -1);

    }

    public int[][] findOptimalAssignment() {

        step1();

        step2();

        step3();

        while (!allColumnsAreCovered()) {

            int[] mainZero = step4();

            while (mainZero == null) {

                step7();

                mainZero = step4();

            }

            if (squareInRow[mainZero[0]] == -1) {

                step6(mainZero);

                step3();

            } else {

                rowIsCovered[mainZero[0]] = 1;

                colIsCovered[squareInRow[mainZero[0]]] = 0;

                step7();

            }

        }

        int[][] optimalAssignment = new int[matrix.length][];

        for (int i = 0; i < squareInCol.length; i++) {

            optimalAssignment[i] = new int[]{i, squareInCol[i]};

        }

        return optimalAssignment;

    }

    private boolean allColumnsAreCovered() {

        for (int i : colIsCovered) {

            if (i == 0) {

                return false;

            }

        }

        return true;

    }

    private void step1() {

        for (int i = 0; i < matrix.length; i++) {

            int currentRowMin = Integer.MAX\_VALUE;

            for (int j = 0; j < matrix[i].length; j++) {

                if (matrix[i][j] < currentRowMin) {

                    currentRowMin = matrix[i][j];

                }

            }

            for (int k = 0; k < matrix[i].length; k++) {

                matrix[i][k] -= currentRowMin;

            }

        }

        for (int i = 0; i < matrix[0].length; i++) {

            int currentColMin = Integer.MAX\_VALUE;

            for (int j = 0; j < matrix.length; j++) {

                if (matrix[j][i] < currentColMin) {

                    currentColMin = matrix[j][i];

                }

            }

            for (int k = 0; k < matrix.length; k++) {

                matrix[k][i] -= currentColMin;

            }

        }

    }

    private void step2() {

        int[] rowHasSquare = new int[matrix.length];

        int[] colHasSquare = new int[matrix[0].length];

        for (int i = 0; i < matrix.length; i++) {

            for (int j = 0; j < matrix.length; j++) {

                if (matrix[i][j] == 0 && rowHasSquare[i] == 0 && colHasSquare[j] == 0) {

                    rowHasSquare[i] = 1;

                    colHasSquare[j] = 1;

                    squareInRow[i] = j;

                    squareInCol[j] = i;

                    continue;

                }

            }

        }

    }

    private void step3() {

        for (int i = 0; i < squareInCol.length; i++) {

            colIsCovered[i] = squareInCol[i] != -1 ? 1 : 0;

        }

    }

    private void step7() {

        int minUncoveredValue = Integer.MAX\_VALUE;

        for (int i = 0; i < matrix.length; i++) {

            if (rowIsCovered[i] == 1) {

                continue;

            }

            for (int j = 0; j < matrix[0].length; j++) {

                if (colIsCovered[j] == 0 && matrix[i][j] < minUncoveredValue) {

                    minUncoveredValue = matrix[i][j];

                }

            }

        }

        if (minUncoveredValue > 0) {

            for (int i = 0; i < matrix.length; i++) {

                for (int j = 0; j < matrix[0].length; j++) {

                    if (rowIsCovered[i] == 1 && colIsCovered[j] == 1) {

                        matrix[i][j] += minUncoveredValue;

                    } else if (rowIsCovered[i] == 0 && colIsCovered[j] == 0) {

                        matrix[i][j] -= minUncoveredValue;

                    }

                }

            }

        }

    }

    private int[] step4() {

        for (int i = 0; i < matrix.length; i++) {

            if (rowIsCovered[i] == 0) {

                for (int j = 0; j < matrix[i].length; j++) {

                    if (matrix[i][j] == 0 && colIsCovered[j] == 0) {

                        staredZeroesInRow[i] = j;

                        return new int[]{i, j};

                    }

                }

            }

        }

        return null;

    }

    private void step6(int[] mainZero) {

        int i = mainZero[0];

        int j = mainZero[1];

        Set<int[]> K = new LinkedHashSet<>();

        K.add(mainZero);

        boolean found = false;

        do {

            if (squareInCol[j] != -1) {

                K.add(new int[]{squareInCol[j], j});

                found = true;

            } else {

                found = false;

            }

            if (!found) {

                break;

            }

            i = squareInCol[j];

            j = staredZeroesInRow[i];

            if (j != -1) {

                K.add(new int[]{i, j});

                found = true;

            } else {

                found = false;

            }

        } while (found);

        for (int[] zero : K) {

            if (squareInCol[zero[1]] == zero[0]) {

                squareInCol[zero[1]] = -1;

                squareInRow[zero[0]] = -1;

            }

            if (staredZeroesInRow[zero[0]] == zero[1]) {

                squareInRow[zero[0]] = zero[1];

                squareInCol[zero[1]] = zero[0];

            }

        }

        Arrays.fill(staredZeroesInRow, -1);

        Arrays.fill(rowIsCovered, 0);

        Arrays.fill(colIsCovered, 0);

    }

}