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 9

**Выполнил:**  студент группы 10701118 Воробей И.А.

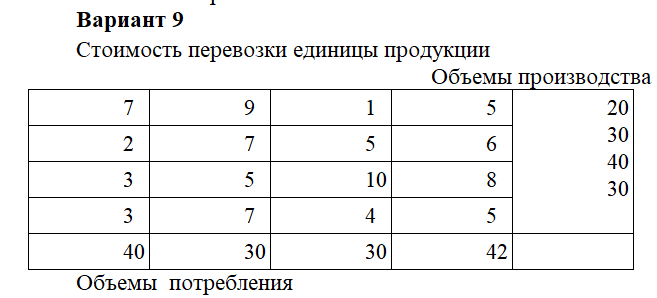
**Приняла ст. преподаватель:** Борисова И.М.

Минск 2020

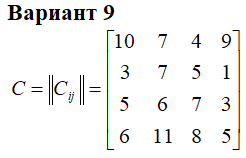
**Цель работы:** изучить способы решения зада дискретного программирования, изучить Венгерский метод и поиск решения в среде MS EXCEL.

**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.*

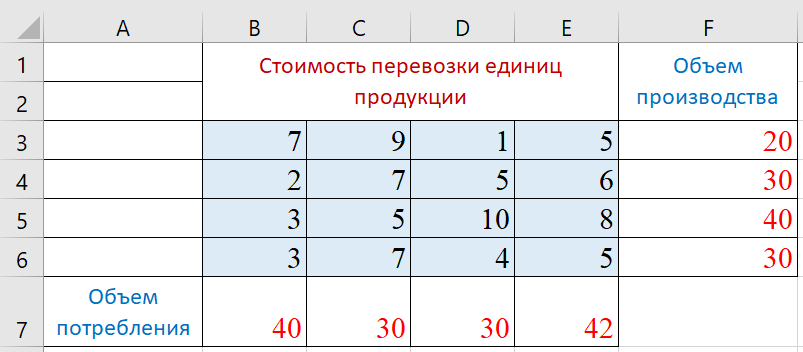


Рисунок 1 – Условия транспортной задачи

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

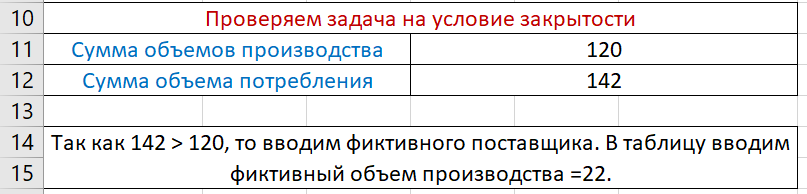


Рисунок 2 – Результат решения задачи

Фиктивный поставщик *Рисунок 3*.



Рисунок 3 – Добавление фиктивного объема производства

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



Рисунок 4 – Результат выполнения *Поиска решения*

Значение целевой функции представлено на *Рисунке 5.*

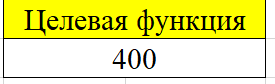
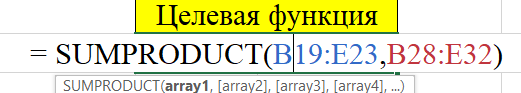


Рисунок 5 – Значение целевой функции

В результате выполнения задачи минимальная стоимость равна 400. План перевозок представлен на *Рисунке 4*.

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

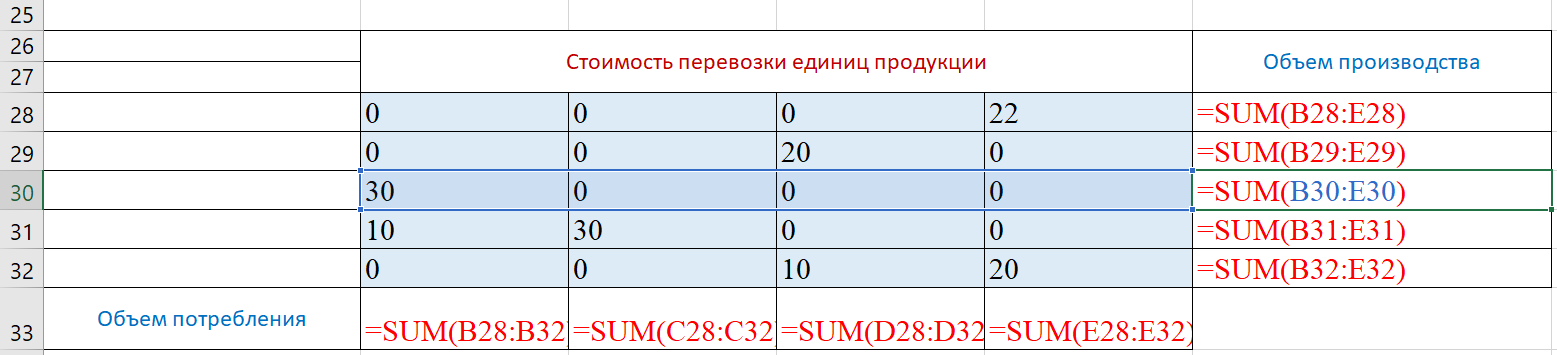
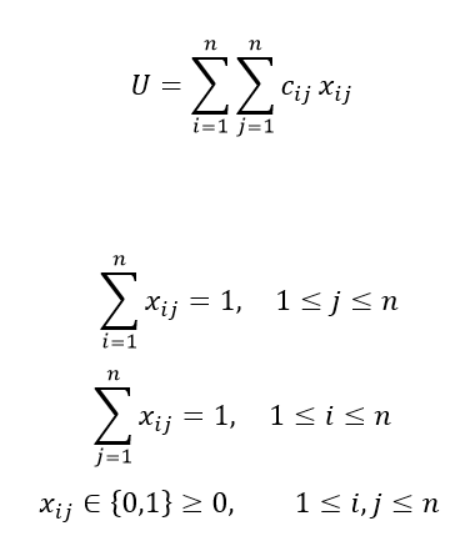
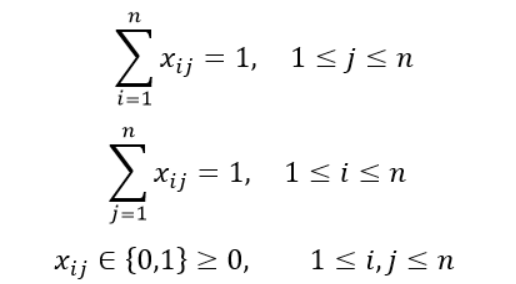


Рисунок 6 – Формулы в таблице Excel.

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

Целевая функция: Ограничения:

 ****

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

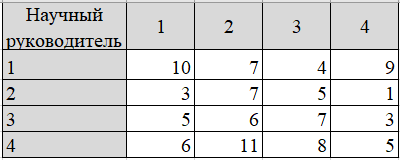


Рисунок 7 – Поиск решения для решения задачи 10 варианта

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

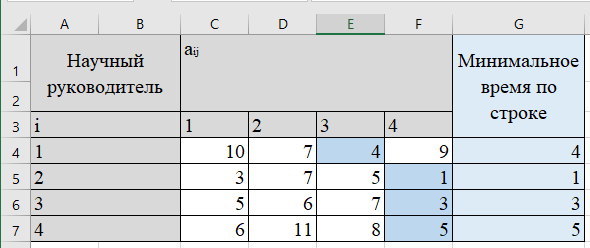


Рисунок 8 – Минимальные значение по строке

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.*

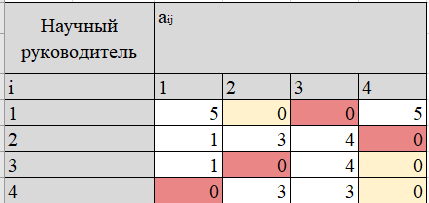


Рисунок 10 – Результат назначений

Проводим поиск допустимого решения, для которого все назначения имеют нулевую стоимость. Определяем матрицу назначения Х, которая позволяет по аналогично расположенным элементам исходной матрицы (в квадратах) вычислить минимальную стоимость назначения. Рисунок 11

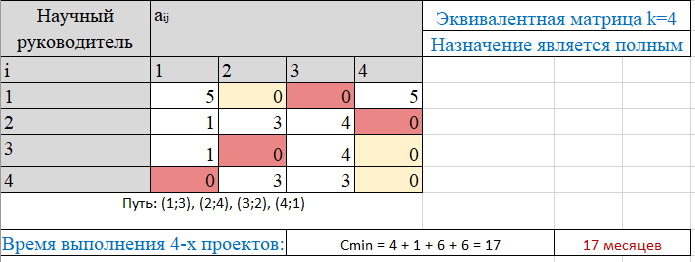


Рисунок 11 – Результат в месяцах.

Перепроверяю через поиск решения. Рузультаты венгерского метода и поиска решения в эксель сошлись. Рисунки 12 - 14

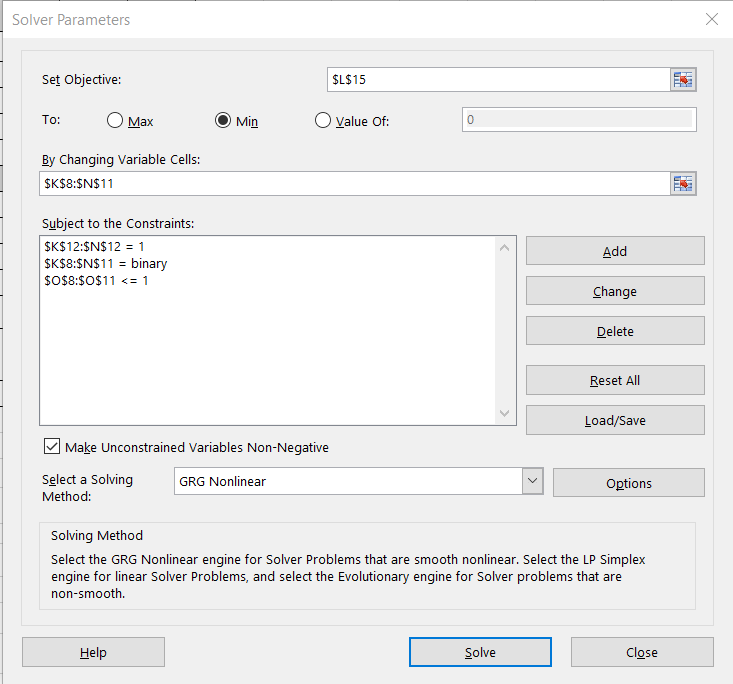


Рисунок 12 – Настройка поиска решений.

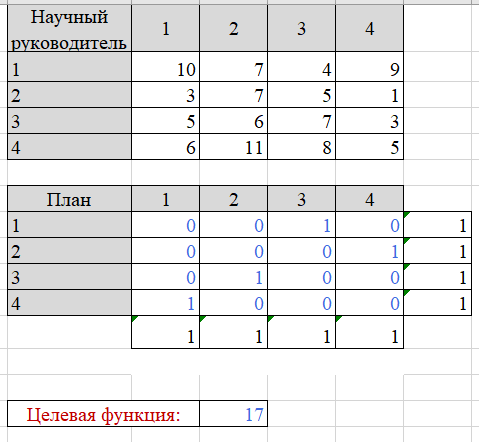


Рисунок 13 – Результат поиска решений.

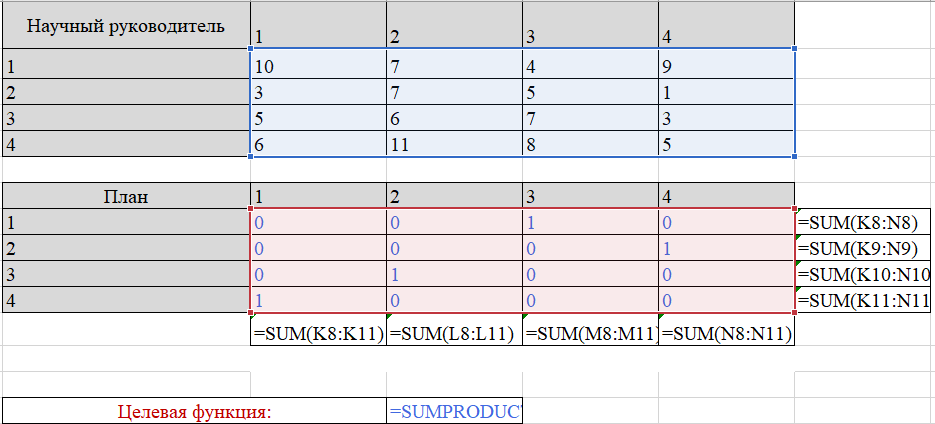


Рисунок 14 – Демонстрация формул.

**Разработка программы для решения задач Венгерским методом**

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 15.*

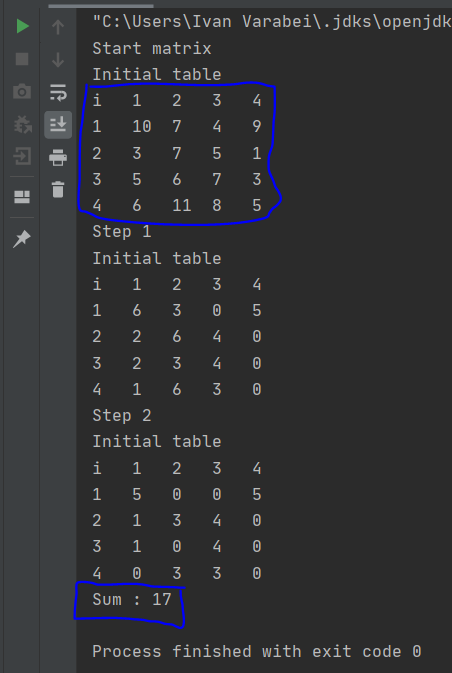


Рисунок 15 – Результат выполнения программы

**Вывод**

* Я решил транспортную задачу используя поиск решения эксель
* Я решил задачу о назначениях венгерским методов, поиском решенияв эксель.
* Реализовал венгерский метод на языка java
* Ответы всех способов решения сошлись.

**Листинг**

**HungarianAlgorithm.java**

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);  
 }  
}

Main.java

public class Main {  
 public static void main(String[] args) {  
  
 int answer = 0;  
  
 int[][] dataMatrix = {  
 {10, 7, 4, 9},  
 {3, 7, 5, 1},  
 {5, 6, 7, 3},  
 {6, 11, 8, 5}  
 };  
 HungarianAlgorithm ha = new HungarianAlgorithm(dataMatrix);  
 int[][] assignment = ha.findOptimalAssignment();  
 }  
}