МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

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

**Отчёт по лабораторной работе №2**

По дисциплине: «Методы и алгоритмы принятия решений»

На тему: «РЕШЕНИЕ ЗАДАЧА ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ»

Вариант 9

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

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

Минск 2020

*Цель работы: изучить симплексный метод решения линейных задач и поиск решения в среде MS EXCEL.*

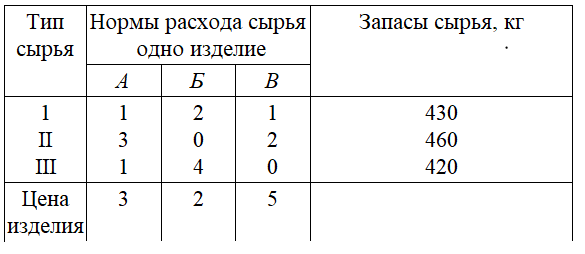
*Постановка задачи:*

*Решить задачу оптимального использования ресурсов на максимум общей стоимости:*

1. *Симплекс методом*
2. *Поиск решения в MS EXCEL*

*Решение и оформление задачи проводить по следующим этапам:*

1. *Математическое описание задачи*
2. *Описание реализации решения задачи с помощью электронных таблиц*



**Выполнение задачи**

1. Составляем форму с данными задачи *Рисунок 1.*
2. Осуществляем абсолютную адресацию к блоку x1 – x3, назначим имя «ПЕРЕМ»



Рисунок 1 – Результат решения задачи

1. Вычислим значение прибыли в ячейки G4(оранжевого цвета *Рисунок 1*). Введем формулу «= SUMPRODUCT(ПЕРЕМ, D4:F4)». И заполняем столбец значений ресурсов, перетаскивая эту ячейку на все остальные(желтые ячейки *Рисунок 1)*. Результаты введения формул представлены на *Рисунке 2.*

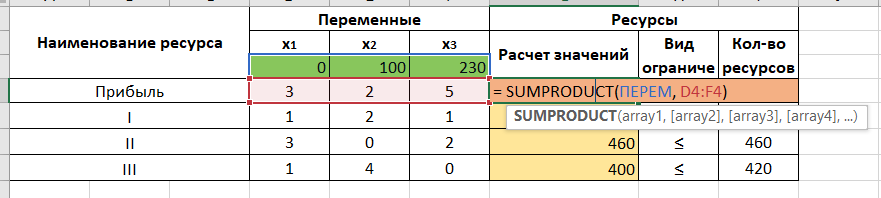


Рисунок 2 – Формулы СУММПРОИЗВ

1. Решаем задачу оптимизации. Выбираем «Поиск решения» и устанавливаем целевую ячейку H4. В поле «Изменяя ячейки переменных» вставляем массив ячеек «ПЕРЕМ», а затем добавляем ограничения на использование ресурсов. Выставляем параметры поиска решения, установив флажки в полях «Линейная модель» и «Неотрицательные значения». Результат настройки окна «Параметры поиска решения» указаны на *Рисунке 3.* Нажимаем кнопку «Найти решения». Результат выполнения *Рисунок 4*.

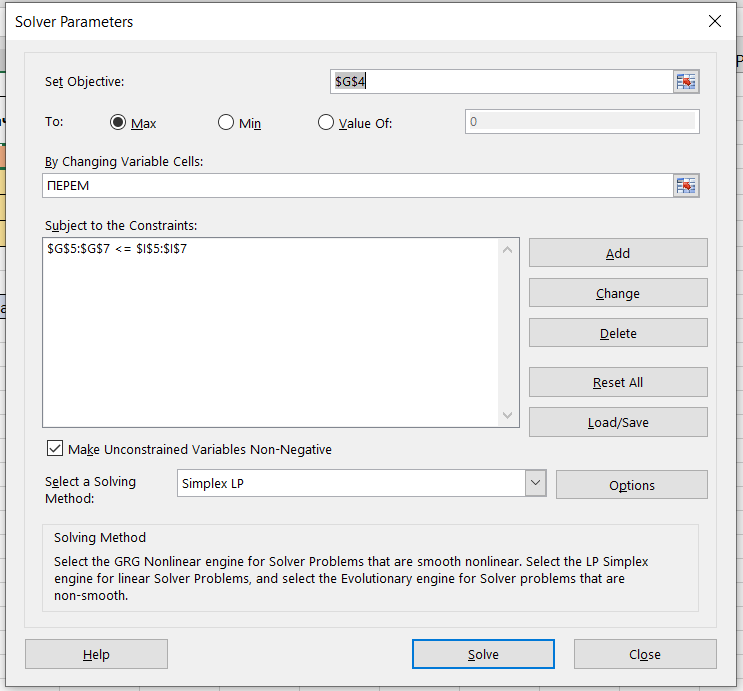


Рисунок 3 – Поиск решения

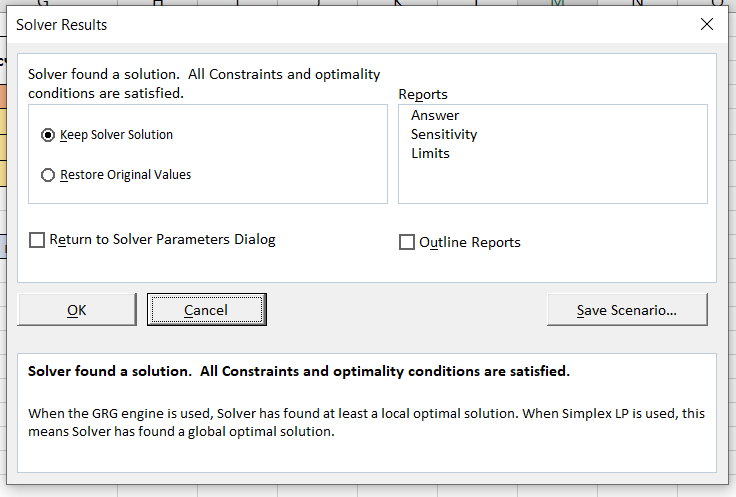




Рисунок 4 – Результат поиска решения

1. **Решение задачи симплекс-методом.**
   1. Для построение опорного плана систему неравенств приведем к системе уравнений путем введения дополнительных переменных (переход к канонической форме) *Рисунок 5.*

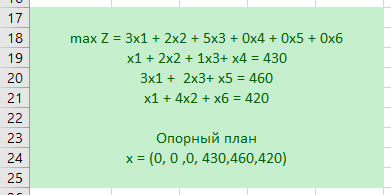


Рисунок 5 – Система уравнений

* 1. Матрицы коэффициентов имеют вид *Рисунок 6.*

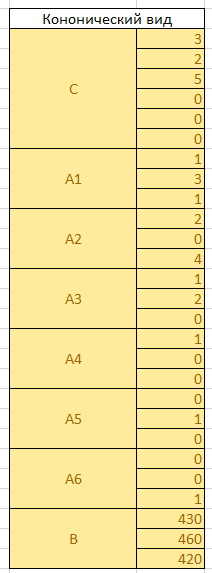


Рисунок 6 – Канонический вид

* 1. Базисное решение называется допустимым, если оно неотрицательно.

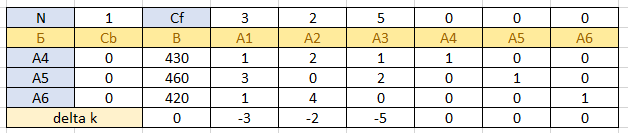


Рисунок 7 – Первоначальная симплекс-таблица

* 1. Нахождение оптимального плана сводится к следующим шагам:

1. Проверка критерия оптимальности.   
2. Определение новой базисной переменной.   
3. Определение новой свободной переменной.

4. Пересчет симплекс-таблицы. 

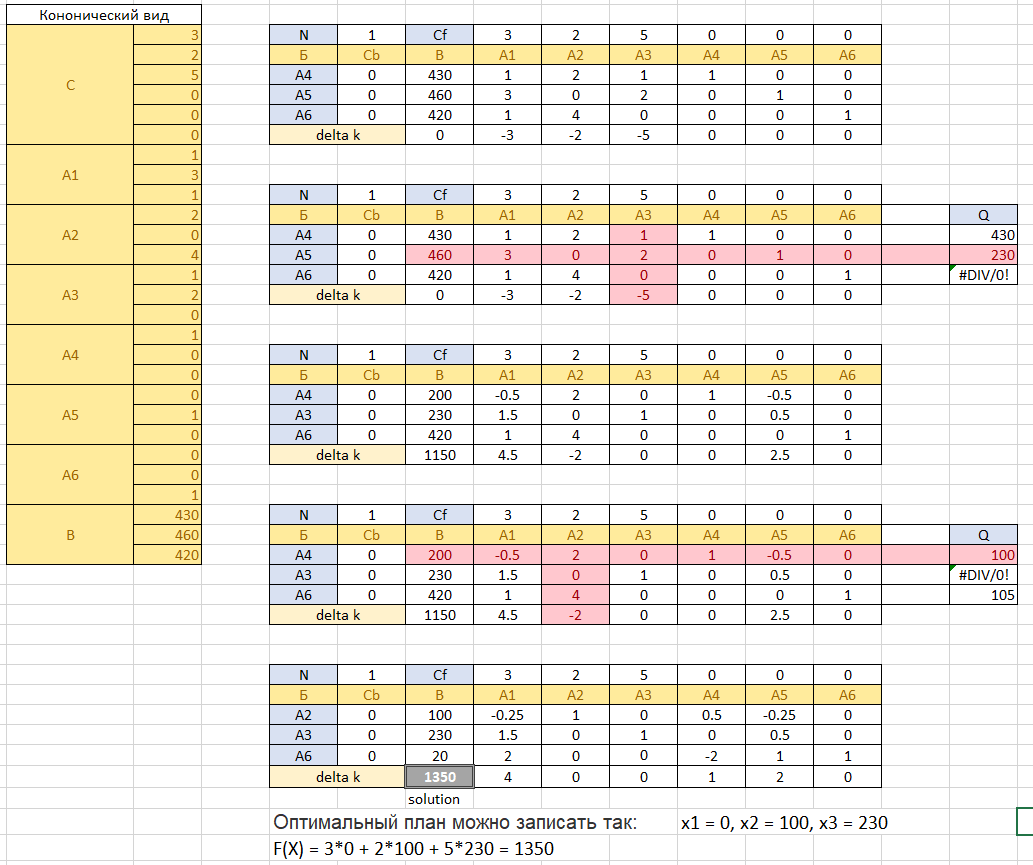


Рисунок 8 – Решение симплекс-методом

Оптимальный план можно записать так:  
x1 = 0, x2 = 100, x3 = 230  
F(X) = 3\*0 + 2\*100 + 5\*230 = 1350

В оптимальный план вошла дополнительная переменная x6. Следовательно, при реализации такого плана имеются недоиспользованные ресурсы 3-го вида в количестве 20.

Значение *4> 0* в столбце а1 означает, что использование x1 - не выгодно.  
Значение *0* в столбце а2 означает, что использование x2 - выгодно.  
Значение *0* в столбце а3 означает, что использование x3 - выгодно.  
Значение 1 в столбце а4 означает, что теневая цена (двойственная оценка) равна y1=1.  
Значение 2 в столбце а5 означает, что теневая цена (двойственная оценка) равна y2=2.  
Значение 0 в столбце а6 означает, что теневая цена (двойственная оценка) равна y3=0.

1. **Программное решение симплекс-методом**

**Программа принимает на вход симплекс таблицу.**

float[][] data = {  
 {1, 2, 1, 1, 0, 0, 430},  
 {3, 0, 2, 0, 1, 0, 460},  
 {1, 4, 0, 0, 0, 1, 420},  
 {-3, -2, -5, 0, 0, 0, 0},  
};

Далее в цикле происходят следующие действия :

1.Проверка критерия оптимальности.

2. Определение новой базисной переменной.

3. Определение новой свободной переменной.

4. Пересчет симплекс-таблицы.

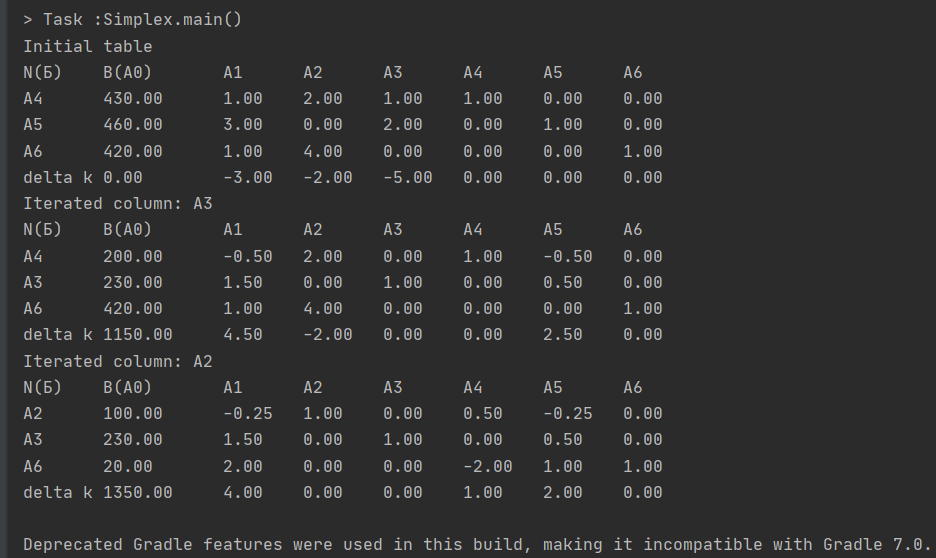
Результаты выполнения программы с нач. Данными ‘data’, представлены на Рисунке 13.

Рисунок 13 – результаты выполнения программы

**Вывод**

Изучил симплексный метод решения линейных задач и поиск решения в среде MS EXCEL. Написал программу решения симплекс методом на языке java.

ЛИСТИНГ ИСХОДНОГО КОДА

import java.util.ArrayList;  
import java.util.List;  
  
public class Simplex {  
 private int rows;  
 private int cols;  
 private float[][] table;  
 private boolean solutionIsUnbounded = false;  
 private int iterations = 0;  
 List<Integer> N = new ArrayList();  
  
 public static void main(String... adf) {  
 float[][] data = {  
 {1, 2, 1, 1, 0, 0, 430},  
 {3, 0, 2, 0, 1, 0, 460},  
 {1, 4, 0, 0, 0, 1, 420},  
 {-3, -2, -5, 0, 0, 0, 0},  
 };  
 Simplex simplex = new Simplex(3, 6);  
 simplex.fillTable(data);  
 System.*out*.println("Initial table");  
 simplex.print();  
 boolean quit = false;  
 while (!quit) {  
 Simplex.ERROR error = simplex.compute();  
 if (error == ERROR.*IS\_OPTIMAL*) {  
 quit = true;  
 }  
 simplex.print();  
 }  
 }  
  
 private void fillN() {  
 for (int i = this.cols - this.rows; i < this.cols; ++i) {  
 this.N.add(i + 1);  
 }  
 }  
  
 public Simplex(int numOfConstraints, int numOfUnknowns) {  
 this.rows = numOfConstraints + 1;  
 this.cols = numOfUnknowns + 1;  
 this.table = new float[this.rows][];  
 this.fillN();  
 for (int i = 0; i < this.rows; ++i) {  
 this.table[i] = new float[this.cols];  
 }  
 }  
  
 public void print() {  
 int i;  
 for (i = 0; i < this.cols + 1; ++i) {  
 if (i == 1) {  
 System.*out*.print("B(A0)\t\t");  
  
 } else if (i == 0) {  
 System.*out*.print("N(Б)\t");  
 } else {  
 System.*out*.print("A" + (i - 1) + "\t\t");  
 }  
 }  
 System.*out*.println();  
 for (i = 0; i < this.rows; ++i) {  
 if (i == this.rows - 1) {  
 System.*out*.print("delta k\t");  
 } else {  
 System.*out*.print("A" + this.N.get(i) + "\t\t");  
 }  
 String valueB = String.*format*("%.2f", this.table[i][this.cols - 1]);  
 System.*out*.print(valueB + "\t\t");  
 for (int j = 0; j < this.cols - 1; ++j) {  
 String value = String.*format*("%.2f", this.table[i][j]);  
 System.*out*.print(value + "\t");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
  
 public void fillTable(float[][] data) {  
 for (int i = 0; i < this.table.length; ++i) {  
 System.*arraycopy*(data[i], 0, this.table[i], 0, data[i].length);  
 }  
 }  
  
 public Simplex.ERROR compute() {  
 if (this.checkOptimality()) {  
 return Simplex.ERROR.*IS\_OPTIMAL*;  
 } else {  
 int pivotColumn = this.findEnteringColumn();  
 System.*out*.println("\t\t\t\t\t\t\tIterated column: A" + (pivotColumn + 1));  
 this.N.set(this.iterations, pivotColumn + 1);  
 ++this.iterations;  
 float[] ratios = this.calculateRatios(pivotColumn);  
 int pivotRow = this.findSmallestValue(ratios);  
 this.formNextTableau(pivotRow, pivotColumn);  
 return Simplex.ERROR.*NOT\_OPTIMAL*;  
 }  
 }  
  
 private void formNextTableau(int pivotRow, int pivotColumn) {  
 float pivotValue = this.table[pivotRow][pivotColumn];  
 float[] pivotRowVals = new float[this.cols];  
 float[] pivotColumnVals = new float[this.cols];  
 float[] rowNew = new float[this.cols];  
 System.*arraycopy*(this.table[pivotRow], 0, pivotRowVals, 0, this.cols);  
  
 int i;  
 for (i = 0; i < this.rows; ++i) {  
 pivotColumnVals[i] = this.table[i][pivotColumn];  
 }  
 for (i = 0; i < this.cols; ++i) {  
 rowNew[i] = pivotRowVals[i] / pivotValue;  
 }  
 for (i = 0; i < this.rows; ++i) {  
 if (i != pivotRow) {  
 for (int j = 0; j < this.cols; ++j) {  
 float c = pivotColumnVals[i];  
 this.table[i][j] -= c \* rowNew[j];  
 }  
 }  
 }  
 System.*arraycopy*(rowNew, 0, this.table[pivotRow], 0, rowNew.length);  
 }  
  
 private float[] calculateRatios(int column) {  
 float[] positiveEntries = new float[this.rows];  
 float[] res = new float[this.rows];  
 int allNegativeCount = 0;  
 int i;  
 for (i = 0; i < this.rows; ++i) {  
 if (this.table[i][column] > 0.0F) {  
 positiveEntries[i] = this.table[i][column];  
 } else {  
 positiveEntries[i] = 0.0F;  
 ++allNegativeCount;  
 }  
 }  
 if (allNegativeCount == this.rows) {  
 this.solutionIsUnbounded = true;  
 } else {  
 for (i = 0; i < this.rows; ++i) {  
 float val = positiveEntries[i];  
 if (val > 0.0F) {  
 res[i] = this.table[i][this.cols - 1] / val;  
 }  
 }  
 }  
 return res;  
 }  
  
 private int findEnteringColumn() {  
 float[] values = new float[this.cols];  
 //int location = false;  
 int count = 0;  
 for (int pos = 0; pos < this.cols - 1; ++pos) {  
 if (this.table[this.rows - 1][pos] < 0.0F) {  
 ++count;  
 }  
 }  
 int location;  
 if (count > 1) {  
 for (int i = 0; i < this.cols - 1; ++i) {  
 values[i] = Math.*abs*(this.table[this.rows - 1][i]);  
 }  
 location = this.findLargestValue(values);  
 } else {  
 location = count - 1;  
 }  
 return location;  
 }  
  
 private int findSmallestValue(float[] data) {  
 int location = 0;  
 float minimum = data[0];  
 for (int c = 1; c < data.length; ++c) {  
 if (data[c] > 0.0F && Float.*compare*(data[c], minimum) < 0) {  
 minimum = data[c];  
 location = c;  
 }  
 }  
 return location;  
 }  
  
 private int findLargestValue(float[] data) {  
 float maximum = 0.0F;  
 int location = 0;  
 maximum = data[0];  
 for (int c = 1; c < data.length; ++c) {  
 if (Float.*compare*(data[c], maximum) > 0) {  
 maximum = data[c];  
 location = c;  
 }  
 }  
 return location;  
 }  
  
 public boolean checkOptimality() {  
 boolean isOptimal = false;  
 int vCount = 0;  
 for (int i = 0; i < this.cols - 1; ++i) {  
 float val = this.table[this.rows - 1][i];  
 if (val >= 0.0F) {  
 ++vCount;  
 }  
 }  
 if (vCount == this.cols - 1) {  
 isOptimal = true;  
 }  
 return isOptimal;  
  
 }  
  
 public float[][] getTable() {  
 return this.table;  
 }  
  
 public static enum ERROR {  
 *NOT\_OPTIMAL*, *IS\_OPTIMAL*,*UNBOUNDED*;  
  
 private ERROR() {  
 }  
 }  
}