Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Факультет информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Курсовая работа по дисциплине
Алгоритмы и вычислительные методы оптимизации

Двойственный симплекс-метод

Вариант 12

Выполнил:

студент гр.ИП-811 _____ / Мироненко К.А. /
ФИО студента

«__» _____ 2021 г.

Проверил

ассистент кафедры ПМиК _____ / Новожилов Д.И. /
ФИО преподавателя

«__» _____ 2021 г. Оценка _____

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи 3
2. Алгоритм двойственного симпекс-метода 5
3. Выполнение поставленной задачи 6
4. Описание основных функций программы 12
Приложение Листинг 13

1. Постановка задачи

1. Перейти к канонической форме записи задачи линейного программирования.

$$Z(x_1, x_2) = p_1 x_1 + p_2 x_2 \rightarrow min$$

$$\begin{cases} a_1 x_1 + a_2 x_2 \geq a \\ b_1 x_1 + b_2 x_2 \geq b \\ c_1 x_1 + c_2 x_2 \geq c \\ x_1;\ x_2 \geq 0 \end{cases}$$

2. Написать программу, решающую задачу линейного программирования в канонической форме (с выводом всех промежуточных таблиц) одним из перечисленных способов (в соответствии с последним столбцом приведенной ниже таблицы):

- симплекс-методом, используя в качестве начальной угловой точки опорное решение с указанными в задании базисными переменными, найденное методом Жордана-Гаусса (1);
- методом искусственного базиса (2);
- двойственным симплекс-методом (3).

3. Решить исходную задачу графически и отметить на чертеже точки, соответствующие симплексным таблицам, полученным при выполнении программы из п.2.

4. Составить двойственную задачу к исходной и найти ее решение на основании теоремы равновесия.

Номер варианта	a	b	c	a_1	b_1	c_1	a_2	b_2	c_2	p_1	p_2	Метод решения задачи
1.	12	33	20	5	5	2	1	4	5	7	1	1 базисные переменные: x_1, x_2, x_3
2.	9	13	16	4	3	2	1	2	5	1	5	2
3.	12	14	68	3	1	4	1	2	11	4	3	3
4.	10	30	42	2	3	3	1	4	8	10	3	1 базисные переменные: x_1, x_2, x_3
5.	30	26	54	5	2	3	3	4	11	2	15	2
6.	33	20	12	5	2	5	4	5	1	8	4	3
7.	11	13	12	4	2	1	1	3	7	7	1	1 базисные переменные: x_1, x_2, x_3
8.	45	8	30	10	1	3	3	1	5	2	10	2
9.	14	13	36	3	2	3	1	1	7	4	3	3
10.	16	9	13	2	4	3	5	1	2	6	1	1 базисные переменные: x_1, x_2, x_4
11.	20	12	33	2	5	5	5	1	4	2	7	2
12.	13	16	9	3	2	4	2	5	1	3	5	3
13.	14	68	12	1	4	3	2	11	1	9	2	1 базисные переменные: x_1, x_2, x_5
14.	42	10	30	3	2	3	8	1	4	2	9	2
15.	26	54	30	2	3	5	4	11	3	2	6	3
16.	36	14	13	3	3	2	7	1	1	6	1	1 базисные переменные: x_1, x_2, x_4
17.	13	12	11	2	1	4	3	7	1	1	11	2
18.	8	30	45	1	3	10	1	5	3	4	5	3
19.	54	30	26	3	5	2	11	3	4	5	2	1 базисные переменные: x_1, x_2, x_4
20.	68	12	14	4	3	1	11	1	2	2	11	2
21.	12	11	13	1	4	2	7	1	3	8	6	3

2. Алгоритм двойственного симпекс-метода

Двойственный симплекс-метод заключается в построении оптимального недопустимого плана с последующим преобразованием его в допустимый, не нарушая оптимальности.

Алгоритм двойственного симплекс-метода:

1) В столбце свободных членов выбирают среди отрицательных минимальный. Это определяет разрешающую строку
2) Для отрицательных элементов разрешающей строки находим симплексные отношения: отношения элементов Z-строки к отрицательным элементам разрешающей строки, взятые по модулю
3) Выбираем минимальное симплексное отношение, соответствующий столбец – разрешающий.
4) Выполняют шаг симплексных преобразований таблицы.
5) Если в столбце свободных членов нет отрицательных, то решение оптимально, иначе переход на п.1.

В случае,

- если в разрешающей строке нет ни одного отрицательного элемента, задача неразрешима
- если в Z-строке найден элемент меньше 0, решение следует продолжить обычным симплекс-методом

3. Выполнение поставленной задачи

1. Перейти к канонической форме записи задачи линейного программирования.

Исходная задача:

$$Z(x_1, x_2) = 3x_1 + 5x_2 \rightarrow min$$

$$\begin{cases} 3x_1 + 2x_2 \geq 13 \\ 2x_1 + 5x_2 \geq 16 \\ 4x_1 + x_2 \geq 9 \\ x_1;\ x_2 \geq 0 \end{cases}$$

Каноническая форма:

$$Z(x_1, x_2) = -3x_1 - 5x_2 \rightarrow max$$

$$\begin{cases} -3x_1 - 2x_2 + x_3 = -13 \\ -2x_1 - 5x_2 + x_4 = -16 \\ -4x_1 - x_2 + x_5 = -9 \\ x_1;\ x_2;\ x_3;\ x_4;\ x_5 \geq 0 \end{cases}$$

2. Написать программу, решающую задачу линейного программирования в канонической форме (с выводом всех промежуточных таблиц) двойственным симплекс-методом.

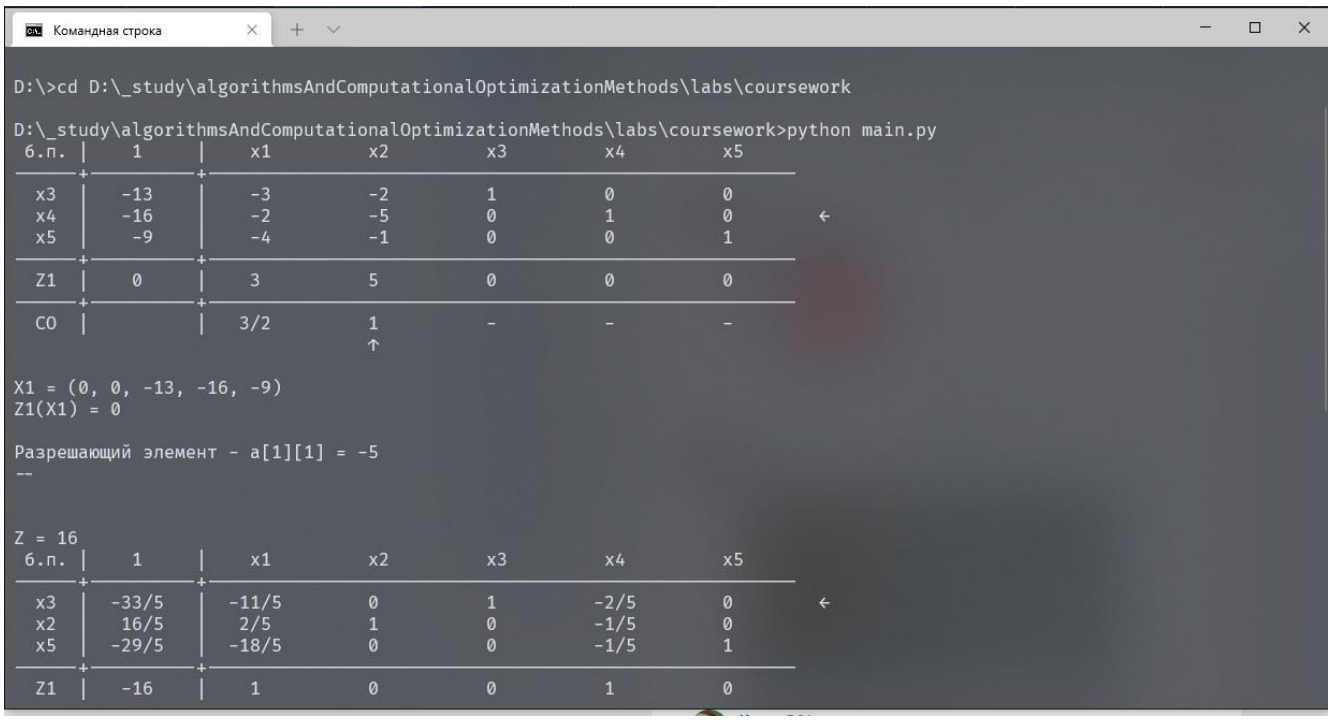

Рис.1.1 Этап программного решения

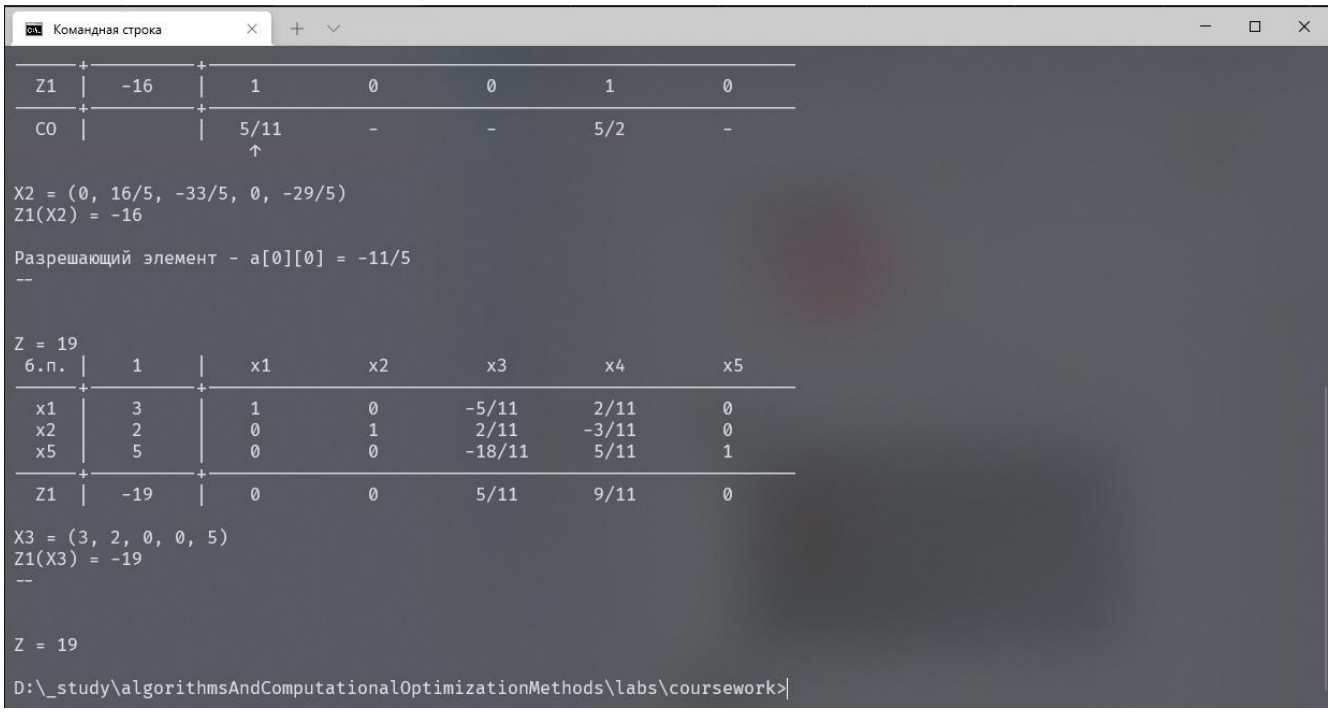

Рис.1.2 Этап программного решения

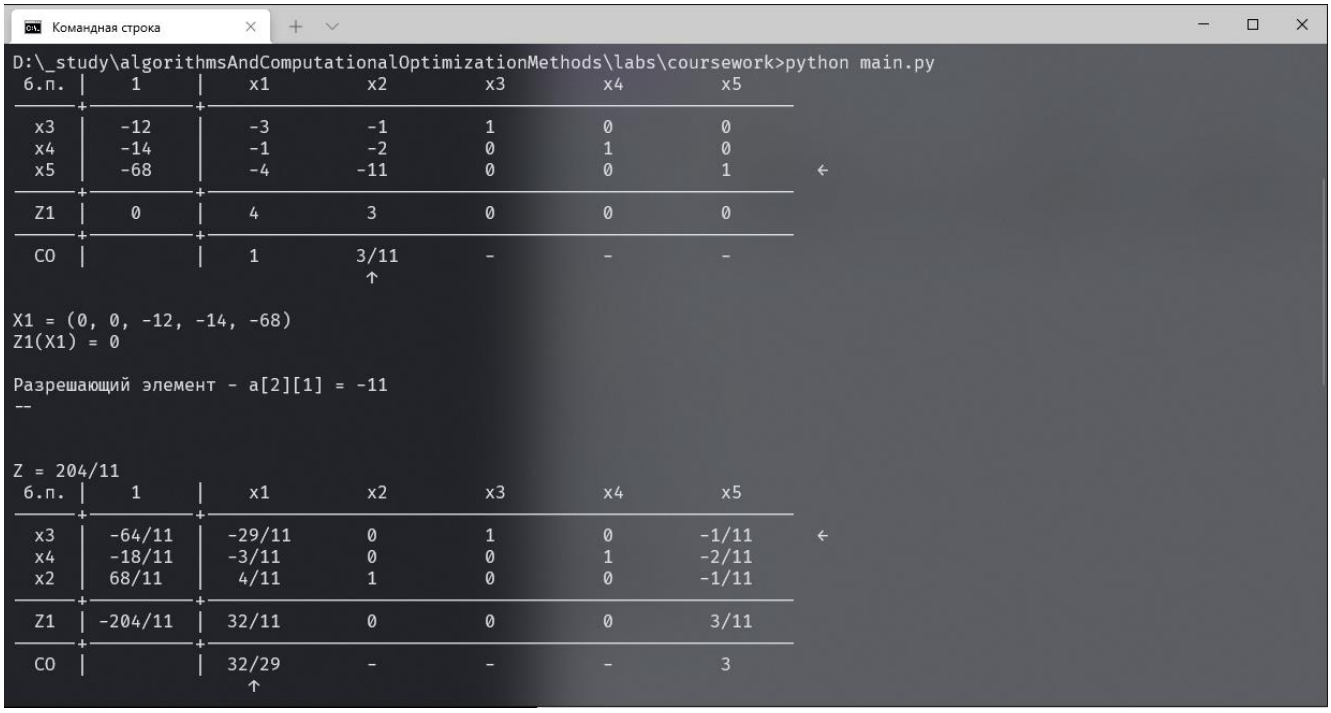

Рис.2.1 Решение другого варианта (Вариант-3)

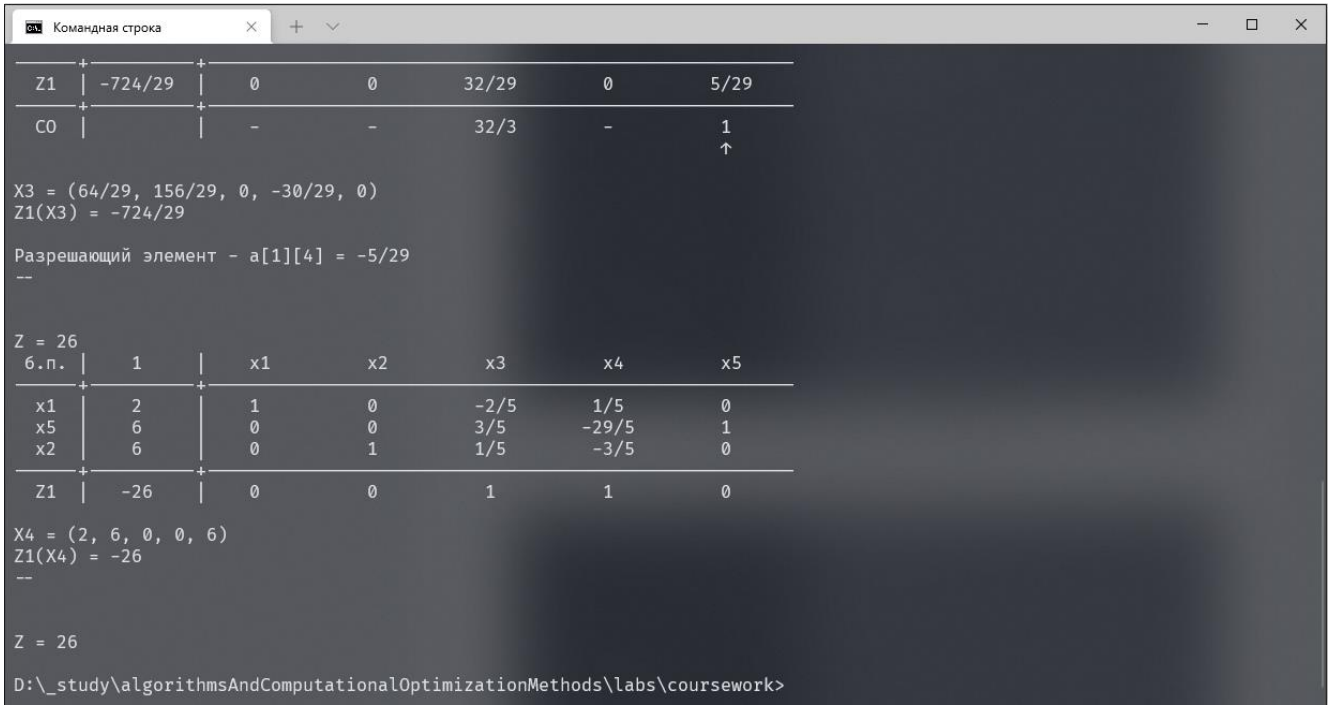

Рис.2.2 Решение другого варианта (Вариант-3)

3. Решить исходную задачу графически и отметить на чертеже точки, соответствующие симплексным таблицам, полученным при выполнении программы из п.2.

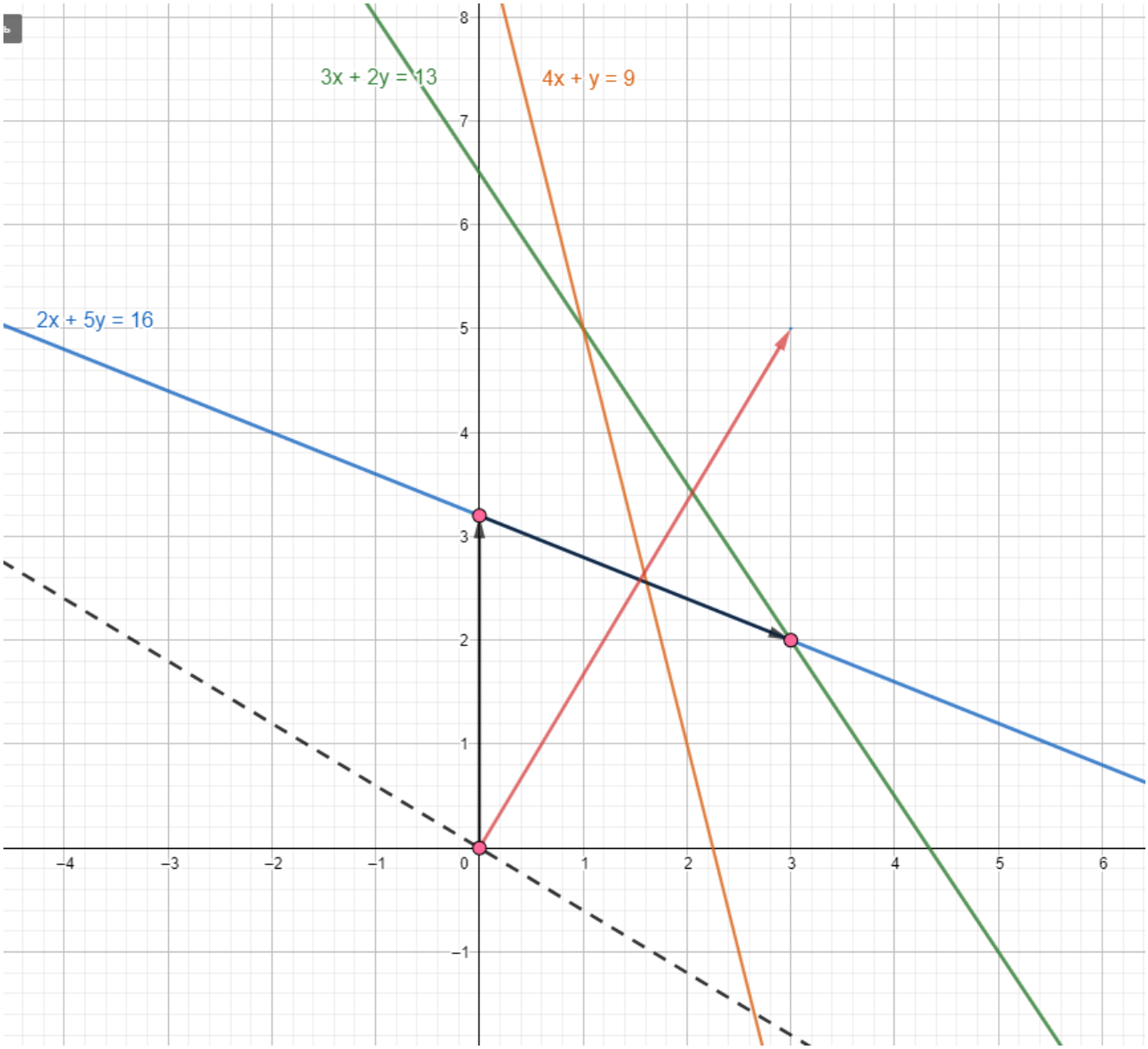

Рис.3 Графическое решение

4. Составить двойственную задачу к исходной и найти ее решение на основании теоремы равновесия.

Исходная задача:

$$Z(x_1, x_2) = 3x_1 + 5x_2 \rightarrow min$$

$$\begin{cases} 3x_1 + 2x_2 \geq 13 \\ 2x_1 + 5x_2 \geq 16 \\ 4x_1 + x_2 \geq 9 \\ x_1;\ x_2 \geq 0 \end{cases}$$

Двойственная задача:

$$\begin{pmatrix} 3 & 2 \\ 2 & 5 \\ 4 & 1 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 3 & 2 & 4 \\ 2 & 5 & 1 \end{pmatrix}$$

Тогда:

$$W(y_1, y_2, y_3) = 13y_1 + 16y_2 + 9y_3 \to max$$

$$\begin{cases} 3y_1 + 2y_2 + 4y_3 \leq 3 \\ 2y_1 + 5y_2 + y_3 \leq 5 \\ y_1, y_2, y_3 \geq 0 \end{cases}$$

Найдем оптимальное решение двойственной задачи по теореме равновесия:

$$\begin{cases} y_1\big(13 - (3x_1 + 2x_2)\big) = 0 \\ y_2\big(16 - (2x_1 + 5x_2)\big) = 0 \\ y_3\big(9 - (4x_1 + x_2)\big) = 0 \\ x_1\big(3 - (3y_1 + 2y_2 + 4y_3)\big) = 0 \\ x_2\big(5 - (2y_1 + 5y_2 + y_3)\big) = 0 \end{cases}$$

Подставим в систему решение $X = (3; 2)$:

$$\begin{cases} y_1\big(13 - (3x_1 + 2x_2)\big) = 0 \\ y_2\big(16 - (2x_1 + 5x_2)\big) = 0 \\ y_3\big(9 - (4x_1 + x_2)\big) = 0 \\ x_1\big(3 - (3y_1 + 2y_2 + 4y_3)\big) = 0 \\ x_2\big(5 - (2y_1 + 5y_2 + y_3)\big) = 0 \end{cases} \Rightarrow \quad \begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ -5y_3 = 0 \\ 3\big(3 - (3y_1 + 2y_2 + 4y_3)\big) = 0 \\ 2\big(5 - (2y_1 + 5y_2 + y_3)\big) = 0 \end{cases} \Rightarrow$$

$$\begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ 3 - (3y_1 + 2y_2 + 4y_3) = 0 \\ 5 - (2y_1 + 5y_2 + y_3) = 0 \end{cases} \Rightarrow \begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ 3y_1 + 2y_2 = 3 \\ 2y_1 + 5y_2 = 5 \end{cases} \Rightarrow$$

$$\begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ y_1 = 1 - \frac{2}{3}y_2 \\ 2(1 - \frac{2}{3}y_2) + 5y_2 = 5 \end{cases} \Rightarrow \begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ y_1 = 1 - \frac{2}{3}y_2 \\ 2(1 - \frac{2}{3}y_2) + 5y_2 = 5 \ | * 3 \end{cases} \Rightarrow$$

$$\begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ y_1 = 1 - \frac{2}{3}y_2 \\ 11y_2 = 9 \end{cases} \Rightarrow \begin{cases} 0y_1 = 0 \\ 0y_2 = 0 \\ y_3 = 0 \\ y_1 = \frac{5}{11} \\ y_2 = \frac{9}{11} \end{cases}$$

$W(y_1, y_2, y_3) = 13y_1 + 16y_2 + 9y_3 \rightarrow max$

Тогда,

$$W_{max} = W\left(\frac{5}{11}, \frac{9}{11}, 0\right) = 13 * \frac{5}{11} + 16 * \frac{9}{11} + 9 * 0 = 19$$

4. Описание основных функций программы

class Fraction – Класс, реализующий дроби

Конструктор:

- *def __init__(self, numerator=0, denominator=1)* – Конструктор класса

Перегрузки:

- *def __add__(self, other)* – Перегрузка сложения 2-х дробей
- *def __sub__(self, other)* – Перегрузка разности 2-х дробей
- *def __mul__self, other)* – Перегрузка произведения 2-х дробей
- *def __ truediv__(self, other)* – Перегрузка частного 2-х дробей
- *def __lt__(self, other)* – Перегрузка оператора «<»
- *def __le__(self, other)* – Перегрузка оператора «<=»
- *def __eq__(self, other)* – Перегрузка оператора «==»
- *def __ne__(self, other)* – Перегрузка оператора «!=»
- *def __gt__(self, other)* – Перегрузка оператора «>»
- *def __ge__(self, other)* – Перегрузка оператора «>=»

Методы:

- *def abs(self)* – Метод, возвращающий абсолютное значение дроби

def read_from_file(filename: str = "input.txt") – Функция, считывающая входные данные из файла

def print_step(matrix, z, x, step_num, simplex_relation=None, resolution_element=None) – Функция, выводящая шаг работы алгоритма

def dual_simplex_method(matrix, z) – Функция, реализующая двойственный симплекс-метод

def main() – функция, реализующая логику работы программы

Приложение Листинг

```
import math
import sys

class Fraction:
    """
    Класс, реализующий дроби
    """
    __slots__ = ('_numerator', '_denominator')

    def __init__(self, numerator=0, denominator=1):
        if type(numerator) is not int or type(denominator) is not int:
            raise TypeError(
                'Fraction(%s, %s) - the numerator and denominator values must be integers' % (numerator,
denominator))
        if denominator == 0:
            raise ZeroDivisionError('Fraction(%s, 0)' % numerator)
        g = math.gcd(numerator, denominator)
        if denominator < 0:
            g = -g
        numerator //= g
        denominator //= g
        self._numerator = numerator
        self._denominator = denominator

    def __add__(self, other):
        """Сумма 2-х дробей"""
        if isinstance(other, Fraction):
            return Fraction(self._numerator * other._denominator + other._numerator * self._denominator,
                            self._denominator * other._denominator)
        return NotImplemented

    def __sub__(self, other):
        """Разность 2-х дробей"""
        if isinstance(other, Fraction):
            return Fraction(self._numerator * other._denominator - other._numerator * self._denominator,
                            self._denominator * other._denominator)
        return NotImplemented

    def __mul__(self, other):
        """Произведение 2-х дробей"""
        if isinstance(other, Fraction):
            return Fraction(self._numerator * other._numerator,
                            self._denominator * other._denominator)
        return NotImplemented

    def __truediv__(self, other):
        """Частное 2-х дробей"""
        if isinstance(other, Fraction):
            return Fraction(self._numerator * other._denominator,
                            self._denominator * other._numerator)
        return NotImplemented

    def __lt__(self, other):
        """x < y"""
```

```
        if isinstance(other, Fraction):
            return self._numerator * other._denominator < other._numerator * self._denominator
        return NotImplemented

    def __le__(self, other):
        """x <= y"""
        if isinstance(other, Fraction):
            return self._numerator * other._denominator <= other._numerator * self._denominator
        return NotImplemented

    def __eq__(self, other):
        """x == y"""
        if isinstance(other, Fraction):
            return self._numerator * other._denominator == other._numerator * self._denominator
        return NotImplemented

    def __ne__(self, other):
        """x != y"""
        if isinstance(other, Fraction):
            return self._numerator * other._denominator != other._numerator * self._denominator
        return NotImplemented

    def __gt__(self, other):
        """x > y"""
        if isinstance(other, Fraction):
            return self._numerator * other._denominator > other._numerator * self._denominator
        return NotImplemented

    def __ge__(self, other):
        """x >= y"""
        if isinstance(other, Fraction):
            return self._numerator * other._denominator >= other._numerator * self._denominator
        return NotImplemented

    def __repr__(self):
        if self._denominator == 1:
            return 'Fraction(%s)' % self._numerator
        else:
            return 'Fraction(%s, %s)' % (self._numerator, self._denominator)

    def __str__(self):
        if self._denominator == 1:
            return str(self._numerator)
        else:
            return '%s/%s' % (self._numerator, self._denominator)

    def abs(self):
        return Fraction(abs(self._numerator), abs(self._denominator))


def read_from_file(filename: str = "input.txt"):
    """
    Функция, считывающая входные данные из файла
    :param filename: имя файла
    :return: Словарь с матрицей("matrix") и z-функцией("z")
    """
```

```
    with open(filename, 'r', encoding="utf-8") as f:
        lines = list(filter(lambda x: x != '' and '#' not in x, list(map(lambda x: x.strip(), f.readlines()))))
    f.close()
    z = list(map(Fraction, map(int, lines[0].split(' '))))
    z = list((i * Fraction(-1))for i in z[:-1]) + z[-1:]
    matrix = list(list(Fraction(int(y)) for y in x.split(' ')) for x in lines[1:])
    # print(z, *matrix, sep='\n')
    return dict(z=z, matrix=matrix)


def print_step(matrix, z, x, step_num, simplex_relation=None, resolution_element=None):
    """
    Функция, выводящая шаг работы алгоритма
    :param matrix: Матрица
    :param z: Z - функция (список)
    :param x: X (список)
    :param step_num: Номер шага (инт)
    :param simplex_relation: Симплексное отношение (список)
    :param resolution_element: Индекс разрешаюшего элемента (словарь(row, col))
    :return: None
    """

    # Шапка
    field_width = 10
    print("{:^6}|".format("б.п."), end='')
    print("{:^{size}}|".format("1", size=field_width), end='')
    for i in range(len(matrix[0]) - 1):
        print("{:^{size}} ".format("x" + str(i + 1), size=field_width), end='')
    print(("\n{:-^6}+{:-^{size}}+{:-^" + str(field_width * (len(matrix[0]) - 1) + len(matrix[0]) - 1) +
"}").format("", "", size=field_width))

    # Строки

    x_index = list()
    for i in range(len(matrix)):
        tmp_flag = False
        for j in range(len(matrix[0]) - 1):
            if matrix[i][j] == Fraction(1):
                for k in range(len(matrix)):
                    if matrix[k][j] == Fraction(0) or k == i:
                        tmp_flag = True
                    else:
                        tmp_flag = False
                        break
            if tmp_flag:
                x_index.append(j)
                break

    for i in range(len(matrix)):
        print("{:^6}|".format("x" + str(x_index[i] + 1)), end='')
        print("{:^{size}}|".format(str(matrix[i][-1]), size=field_width), end='')
        for j in range(len(matrix[0]) - 1):
            print("{:^{size}} ".format(str(matrix[i][j]), size=field_width), end='')
        # стрелочка ←
        if resolution_element and i == resolution_element["row"]:
            print("  ←", end="")
```

```
        print()
    print(("{:-^6}+{:-^{size}}+{:-^" + str(field_width * (len(matrix[0]) - 1) + len(matrix[0]) - 1) + "}").format(",
", ", size=field_width))

    # Z
    print("{:^6}|".format("Z1"), end='')
    print("{:^{size}}|".format(str(z[-1]), size=field_width), end='')
    for i in range(len(z) - 1):
        print("{:^{size}} ".format(str(z[i]), size=field_width), end='')

    # CO
    if simplex_relation:
        print(("\n{:-^6}+{:-^{size}}+{:-^" + str(field_width * (len(matrix[0]) - 1) + len(matrix[0]) - 1) +
"}").format(", ", ", size=field_width))
        print("{:^6}|".format("CO"), end='')
        print("{:^{size}}|".format(", size=field_width), end='')
        for i in range(len(simplex_relation)):
            if simplex_relation[i] != Fraction(sys.maxsize):
                print("{:^{size}} ".format(str(simplex_relation[i]), size=field_width), end='')
            else:
                print("{:^{size}} ".format("-", size=field_width), end='')
    print()
    # стрелочка ↑
    if resolution_element:
        print(("{:^6} {:^{size}} {:^" + str(field_width * (resolution_element["col"]) + resolution_element["col"])
+ "}{:^{size}}").format(", ", ", "↑", size=field_width))
    print()

    # X
    print("X" + str(step_num) + " = " + "(", end='')
    print(*x, sep=', ', end='')
    print(")")

    # Z(X)

    print("Z1(X" + str(step_num) + ") = " + str(z[-1]))


step = 0


def dual_simplex_method(matrix, z):
    global step
    flag = True
    while flag:
        step += 1
        flag = any(x < Fraction(0) for x in [x[-1] for x in matrix])
        res = list(Fraction(0) for _ in range(len(matrix[0]) - 1))

        for i in range(len(matrix)):
            tmp_flag = False
            for j in range(len(matrix[0]) - 1):
                if matrix[i][j] == Fraction(1):
                    for k in range(len(matrix)):
                        if matrix[k][j] == Fraction(0) or k == i:
                            tmp_flag = True
```

```
                else:
                    tmp_flag = False
                    break
            if tmp_flag:
                res[j] = matrix[i][-1]
                break

    if flag:
        if any(x < Fraction(0) for x in z[:-1]):
            print("Отрицательный элемент в Z строке. \nДальше необходимо решать простым симплекс методом")
            flag = False
            break

        # Поиск разрешающей строки
        b = list(x[-1] for x in matrix)
        negative_b = list(filter(lambda q: q < Fraction(0), b))
        resolution_row = b.index(min(negative_b))

        if not any(x < Fraction(0) for x in matrix[resolution_row][:-1]):
            print("Нет решений.\nВ разрешающей строке нет отрицательных элементов")
            flag = False
            break

        # Поиск разрешающего столбца
        simplex_relation = list(Fraction(sys.maxsize) for i in range(len(matrix[0]) - 1))
        for i in range(len(matrix[0]) - 1):
            if matrix[resolution_row][i] < Fraction(0):
                simplex_relation[i] = z[i].abs() / matrix[resolution_row][i].abs()
        resolution_col = simplex_relation.index(min(simplex_relation))

        print_step(matrix, z, res, step, simplex_relation, dict(row=resolution_row, col=resolution_col))

        print("\nРазрешающий элемент - a[{}][{}] = {}".format(resolution_row, resolution_col, matrix[resolution_row][resolution_col]))

        tmp = matrix[resolution_row][resolution_col]
        for i in range(len(matrix[resolution_row])):
            matrix[resolution_row][i] /= tmp

        for i in range(len(matrix)):
            if i != resolution_row and matrix[i][resolution_col] != Fraction(0):
                coeff = matrix[i][resolution_col] * Fraction(-1)
                for j in range(len(matrix[0])):
                    matrix[i][j] = matrix[i][j] + matrix[resolution_row][j] * coeff

        coeff = z[resolution_col] * Fraction(-1)
        for i in range(len(z)):
            z[i] = z[i] + matrix[resolution_row][i] * coeff

    else:
        print_step(matrix, z, res, step)

    print("--\n\n")

    print("Z = " + str(z[-1].abs()))
```

```
def main():
    # Чтение из файла
    data = read_from_file()
    z, matrix = data["z"], data["matrix"]
    # print_step(matrix, z, list(Fraction(i) for i in range(len(matrix[0]))), 0, list(0 for i in range(len(matrix[0]) -
1)), dict(row=2, col=3))

    dual_simplex_method(matrix, z)
    return


if __name__ == '__main__':
    main()
```