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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

[1. Постановка задачи 3](#_Toc73396368)

[2. Алгоритм двойственного симпекс-метода 5](#_Toc73396369)

[3. Выполнение поставленной задачи 6](#_Toc73396370)

[4. Описание основных функций программы 12](#_Toc73396371)

[*Приложение* Листинг 13](#_Toc73396372)

# Постановка задачи

1. Перейти к канонической форме записи задачи линейного программирования.
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).

1. Решить исходную задачу графически и отметить на чертеже точки, соответствующие симплексным таблицам, полученным при выполнении программы из п.2.
2. Составить двойственную задачу к исходной и найти ее решение на основании теоремы равновесия.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер варианта** | ***а*** | ***b*** | ***с*** | ***а*1** | ***b*1** | ***с*1** | ***а*2** | ***b*2** | ***с*2** | ***p*1** | ***p*2** | **Метод решения задачи** |
| **1.** | 12 | 33 | 20 | 5 | 5 | 2 | 1 | 4 | 5 | 7 | 1 | 1  базисные переменные: |
| **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: |
| **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: |
| **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: |
| **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: |
| **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: |
| **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: |
| **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 |

# 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

# Выполнение поставленной задачи

1. Перейти к канонической форме записи задачи линейного программирования.

*Исходная задача:*

*Каноническая форма:*

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

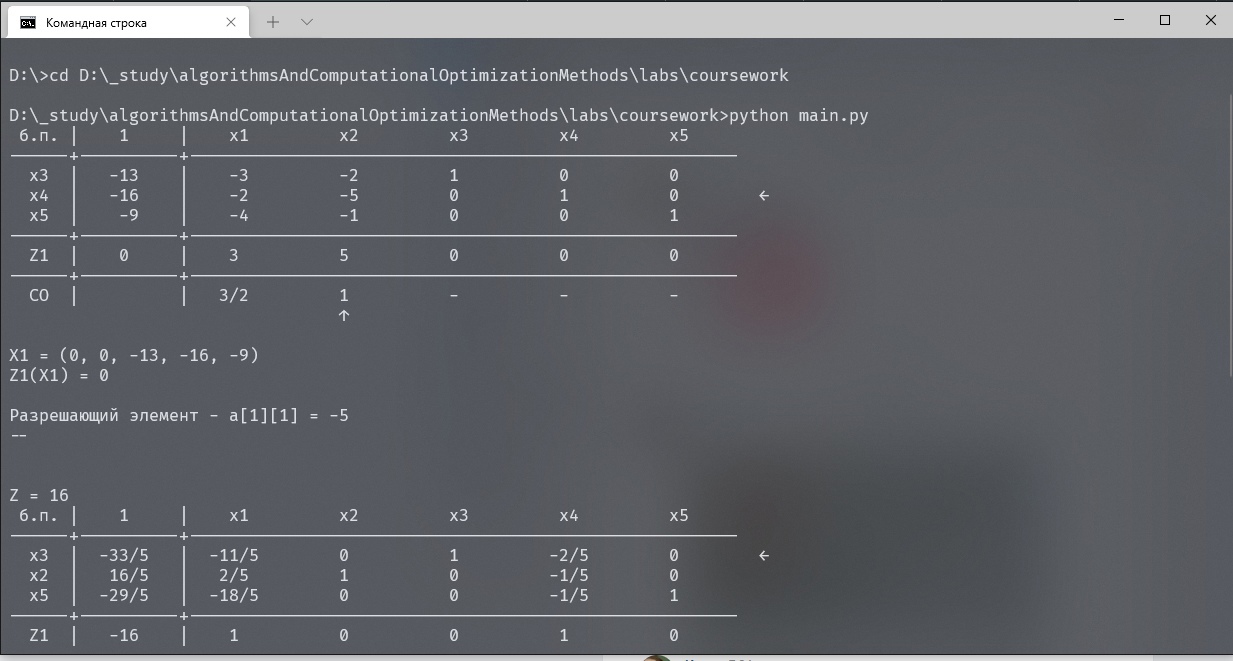


Рис.1.1 Этап программного решения

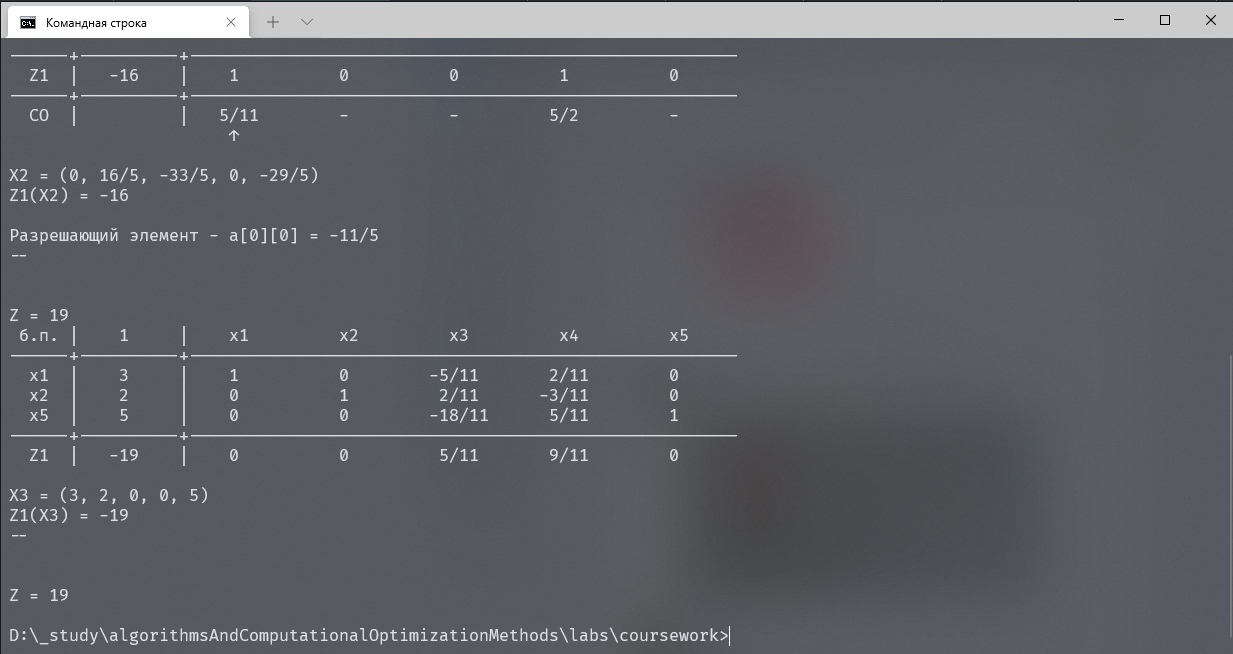


Рис.1.2 Этап программного решения

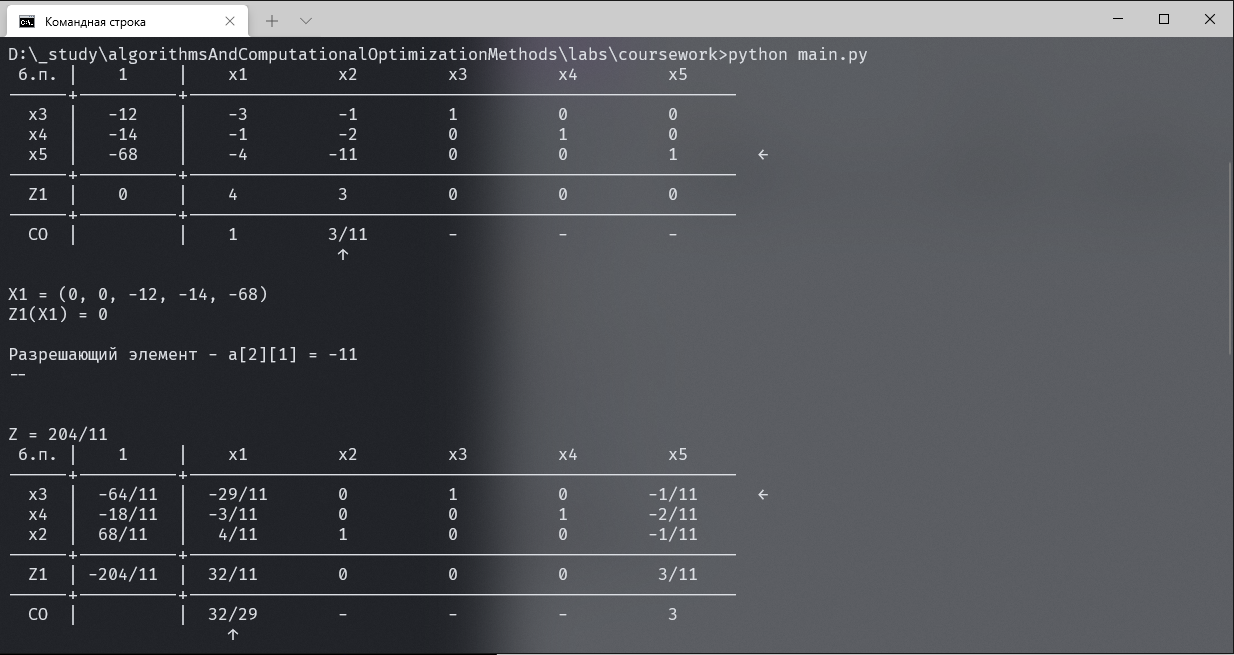


Рис.2.1 Решение другого варианта (Вариант-3)

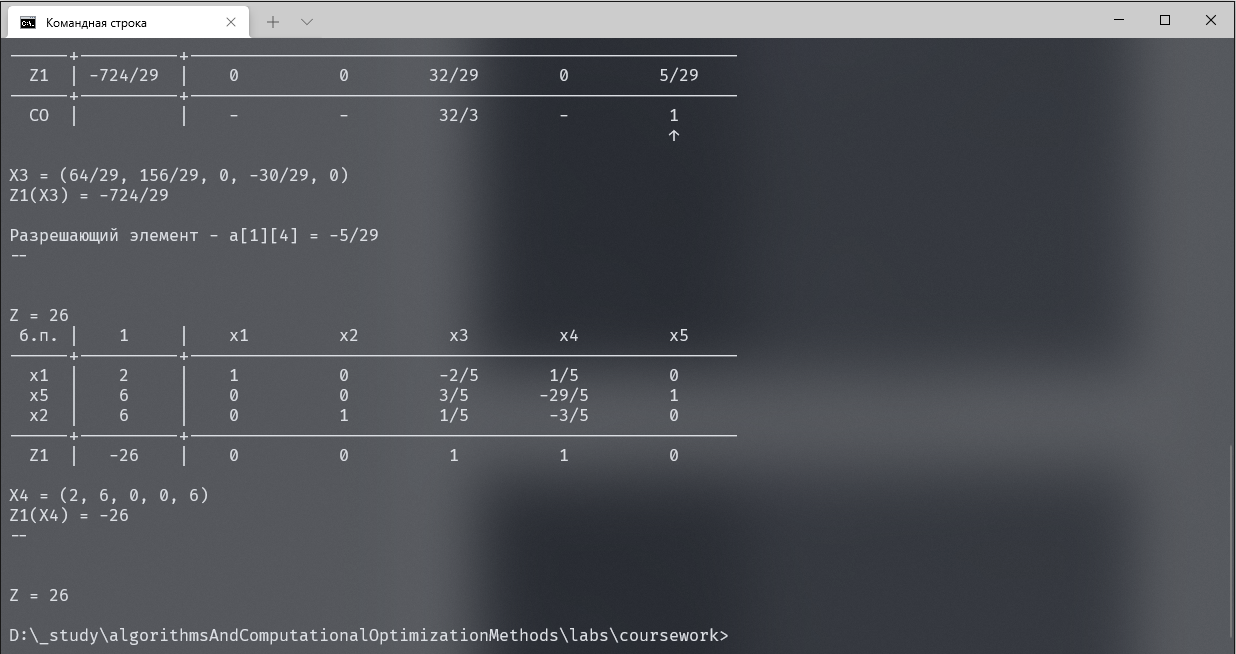


Рис.2.2 Решение другого варианта (Вариант-3)

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

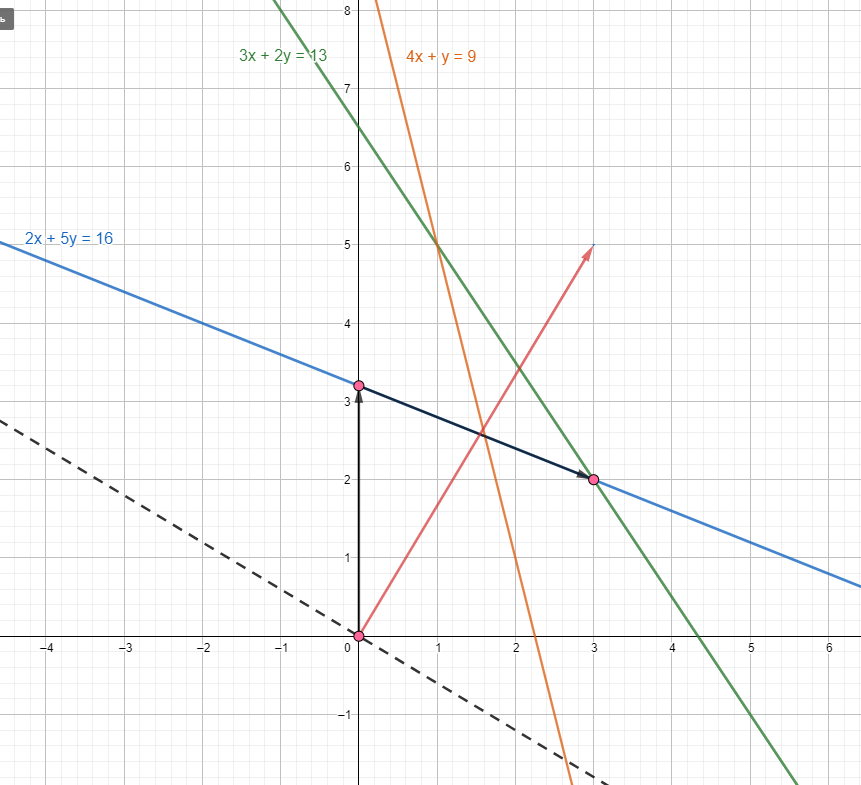


Рис.3 Графическое решение

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

*Исходная задача:*

*Двойственная задача:*

*Тогда:*

Найдем оптимальное решение двойственной задачи по теореме равновесия:

Подставим в систему решение :

*Тогда,*

# 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='')

# СО

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("СО"), 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()