Министерство образования Российской Федерации

МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.БАУМАНА

Факультет: Информатика и системы управления (ИУ)
Кафедра: Информационная безопасность (ИУ8)

МЕТОДЫ ОПТИМИЗАЦИИ

Лабораторная работа №3 на тему:

«Целочисленное линейное программирование.

Метод ветвей и границ»

Вариант 1

Преподаватель:

Коннова Н.С.

Студент:

Александров А.Н.

Группа:

ИУ8-34

Москва, 2020

Цель работы:

изучить постановку задачи целочисленного линейного программирования (ЦЛП); овладеть навыками решения задач ЦЛП с помощью метода ветвей и границ (МВГ).

Постановка задачи:

Требуется найти решение следующей задачи:

$$F = cx \to max,$$

$$Ax \leq b,$$

$$x_1, x_2, x_3 \geq 0$$

Необходимо среди всех *3*-мерных векторов $\mathbf{x} = (x_1, x_2, x_3),\ x_i \in \mathrm{Z}_+^0; \mathrm{x_i} \geq 0, \mathrm{i} = 1, 2, 3$, удовлетворяющих системе

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \leq b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \leq b_3 \end{cases};$$

такой, для которого достигается минимум ЦФ:

$$\min F = cx_1 + c_2x_2 + c_3x_3.$$

Здесь:

c – вектор коэффициентов целевой функции(ЦФ),

A – матрица системы ограничений,

b – вектор правой части системы ограничений.

$$c = (5\ 6\ 4)$$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 0 \\ 0 & 0.5 & 4 \end{pmatrix}$$

$$b = \begin{pmatrix} 7 \\ 8 \\ 6 \end{pmatrix}$$

Ход работы:

Исходная задача ЦЛП имеет вид:

$$F = 5x_1 + 6x_2 + 4x_3 \rightarrow max;$$

$$\begin{cases} x_1 + x_2 + x_3 \leq 7 \\ x_1 + 3x_2 \leq 8 \\ 0.5x_2 + 4x_3 \leq 8 \end{cases};$$

$$x_1, x_2, x_3 \in \mathrm{Z}_+^0$$

Оптимальное решение задачи ЛП, полученное мной в лабораторной работе

$$x_1 = 6.5,\ x_2 = 0.5,\ x_3 = 0; F = 35.5.$$

Всего имеется 33 допустимых целочисленных решения:

(0, 0, 0);	(1, 0, 0);	(2, 0, 0);	(3, 0, 0);	(4, 0, 0);	(5, 0, 0);	(6, 0, 0);	(7, 0, 0)
(0, 0, 1);	(1, 0, 1);	(2, 0, 1);	(3, 0, 1);	(4, 0, 1);	(5, 0, 1);	(6, 0, 1);	
(0, 1, 0);	(1, 1, 0);	(2, 1, 0);	(3, 1, 0);	(4, 1, 0);	(5, 1, 0);		
(0, 1, 1);	(1, 1, 1);	(2, 1, 1);	(3, 1, 1);	(4, 1, 1);	(5, 1, 1);		
(0, 2, 0);	(1, 2, 0);	(2, 2, 0);					
(0, 2, 1);	(1, 2, 1);	(2, 2, 1);					

Полный перебор всех вариантов даёт решение:

$$x_1 = 5,\ x_2 = 1,\ x_3 = 1; F = 35.$$

Для решения задачи ЦЛП методом ветвей и границ воспользовался дополненной версией собственной программы, написанной на языке программирования **Python**. Немного о дополнительных файлах проекта (см. Приложение А):

1. brute_force_method.py

В данном файле содержится класс ***BruteForceMethod***. Он реализует полный перебор всех целочисленных решений и нахождение лучшего среди них.

Поля класса ***BruteForceMethod***:

Поле **obj_func_coffs_**: хранит коэффициенты целевой функции (ЦФ).

Поле **obj_constraint_system_lhs_**: хранит левую часть системы ограничений.

Поле **obj_constraint_system_rhs_**: хранит правую часть системы ограничений.

Поле **func_direction_**: хранит в себе направление целевой функции (поиск минимума или поиск максимума).

Поле **all_solutions_**: заполняется списком всевозможных значений аргументов, удовлетворяющих ограничениям.

Поля **max_ind_, max_func_value_**: индекс максимального в списке **all_solutions_** и само значение функции соответственно.

2. branch_and_bound.py

В данном файле реализован класс ***BranchAndBound***, отвечающий за решение задачи ЦЛП методом ветвей и границ.

Поля класса ***BranchAndBound***:

Поле **obj_func_coffs_**: хранит коэффициенты целевой функции (ЦФ).

Поле **obj_constraint_system_lhs_**: хранит левую часть системы ограничений.

Поле **obj_constraint_system_rhs_**: хранит правую часть системы ограничений.

Поле **func_direction_**: хранит в себе направление целевой функции (поиск минимума или поиск максимума).

Поле **simplex_table_**: хранит в себе начальную симплекс таблицу.

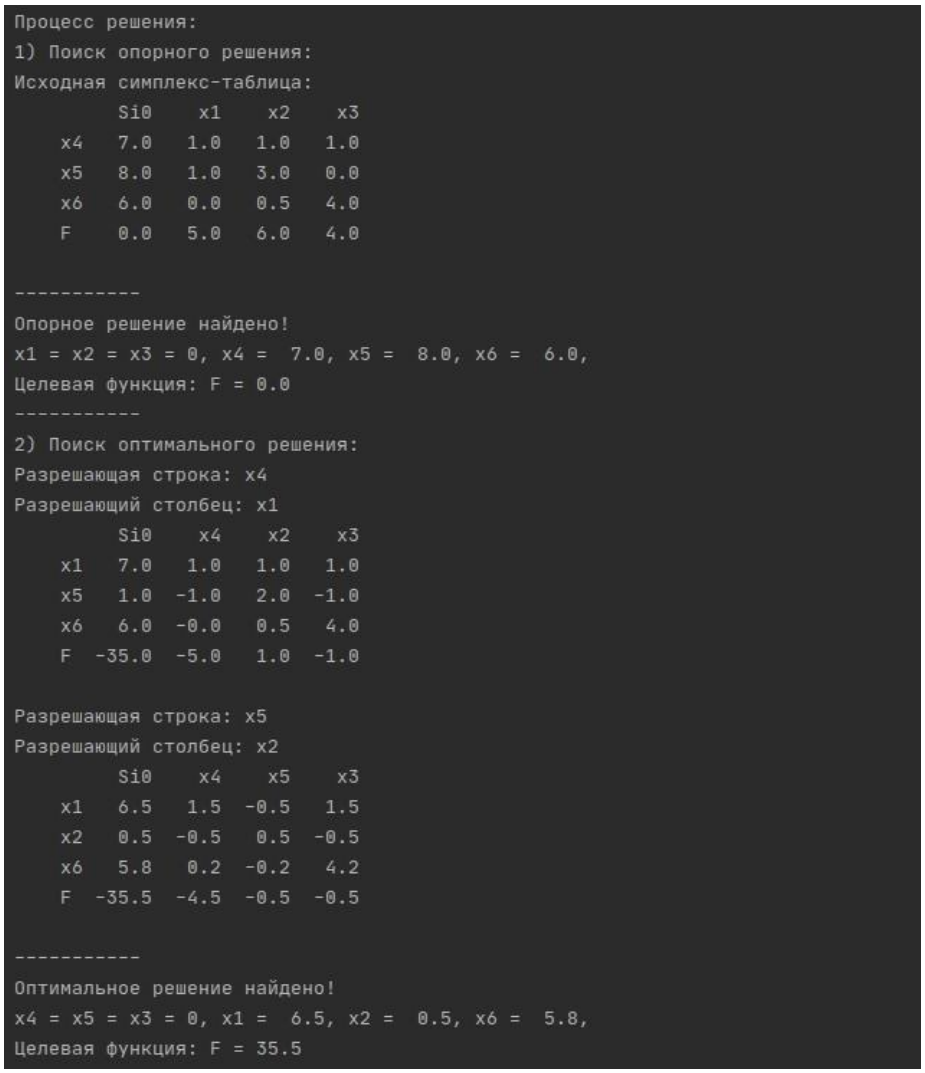
Поле **extra_constraints_lhs_**: дополнительные ограничения, добавляемые в ходе ветвления по дробным переменным.

Поле **extra_constraints_rhs_**: дополнительные ограничения, добавляемые в ходе ветвления по дробным переменным.

Поле **solutions_storage_**: хранилище целочисленных решений, полученных в ходе решения задачи методом ветвей и границ.

Решим исходную задачу ЦЛП:

Рисунок 1 Решение задачи ЛП симплекс-методом.



```
Процесс решения:
1) Поиск опорного решения:
Исходная симплекс-таблица:
         Si0    x1    x2    x3
    x4   7.0   1.0   1.0   1.0
    x5   8.0   1.0   3.0   0.0
    x6   6.0   0.0   0.5   4.0
    F    0.0   5.0   6.0   4.0

-----------
Опорное решение найдено!
x1 = x2 = x3 = 0, x4 =  7.0, x5 =  8.0, x6 =  6.0,
Целевая функция: F = 0.0
-----------
2) Поиск оптимального решения:
Разрешающая строка: x4
Разрешающий столбец: x1
         Si0    x4    x2    x3
    x1   7.0   1.0   1.0   1.0
    x5   1.0  -1.0   2.0  -1.0
    x6   6.0  -0.0   0.5   4.0
    F  -35.0  -5.0   1.0  -1.0

Разрешающая строка: x5
Разрешающий столбец: x2
         Si0    x4    x5    x3
    x1   6.5   1.5  -0.5   1.5
    x2   0.5  -0.5   0.5  -0.5
    x6   5.8   0.2  -0.2   4.2
    F  -35.5  -4.5  -0.5  -0.5

-----------
Оптимальное решение найдено!
x4 = x5 = x3 = 0, x1 =  6.5, x2 =  0.5, x6 =  5.8,
Целевая функция: F = 35.5
```

Решение задачи ЛП:

$$x_1 = 6.5,\ x_2 = 0.5,\ x_3 = 0; F = 35.5.$$

Осуществим ветвление по переменной x_1.

1)Введём новое ограничение $x_1 \leq 6$ и решим задачу ЦЛП:

$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 + x_7 = 6 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \in \mathrm{Z}_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 2 Исходная симплекс-таблица.

```
Исходная симплекс-таблица:
         Si0    x1    x2    x3
    x4   7.0   1.0   1.0   1.0
    x5   8.0   1.0   3.0   0.0
    x6   6.0   0.0   0.5   4.0
    x7   6.0   1.0   0.0   0.0
    F    0.0   5.0   6.0   4.0

-----------
Опорное решение найдено!
x1 = x2 = x3 = 0, x4 =  7.0, x5 =  8.0, x6 =  6.0, x7 =  6.0,
Целевая функция: F = 0.0
-----------
```

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 3 Конечная симплекс-таблица

```
         Si0    x7    x5    x4
    x3   0.3  -0.7  -0.3   1.0
    x2   0.7  -0.3   0.3  -0.0
    x6   4.3   2.8   1.2  -4.0
    x1   6.0   1.0   0.0  -0.0
    F  -35.3  -0.3  -0.7  -4.0

-----------
Оптимальное решение найдено!
x7 = x5 = x4 = 0, x3 =  0.3, x2 =  0.7, x6 =  4.3, x1 =  6.0,
Целевая функция: F = 35.3
-----------
```

Полученное решение $x_1 = 6,\ x_2 = 0.7,\ x_3 = 0.3; F = 35.5$. целочисленным не является.

Осуществим ветвление по переменной x_3.

1.1)Введём новое ограничение $x_3 \leq 0$ и получим задачу ЦЛП:

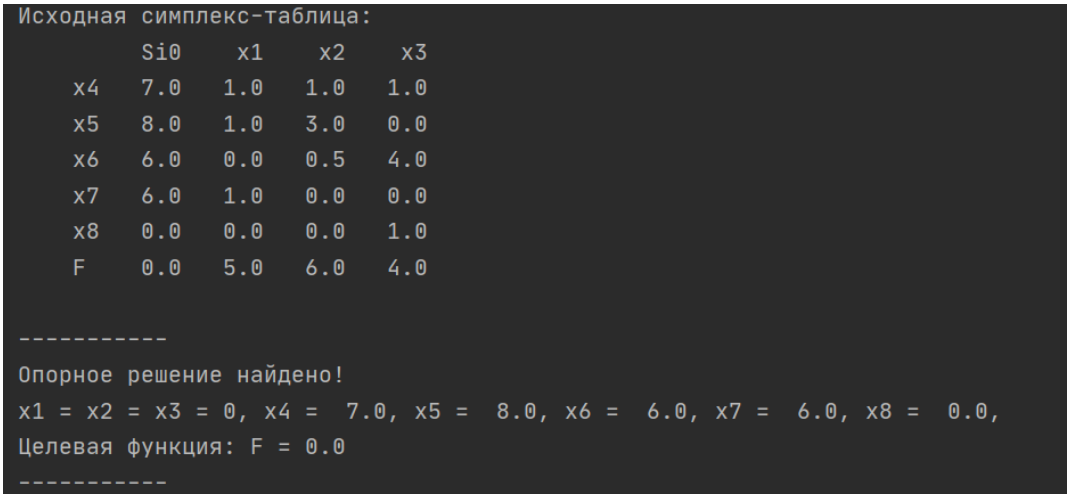
$$F = -5x_1 - 6x_2 - 4x_3 \to min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 + x_7 = 6 \\ x_3 + x_8 = 0 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in Z_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 4 Исходная симплекс-таблица.

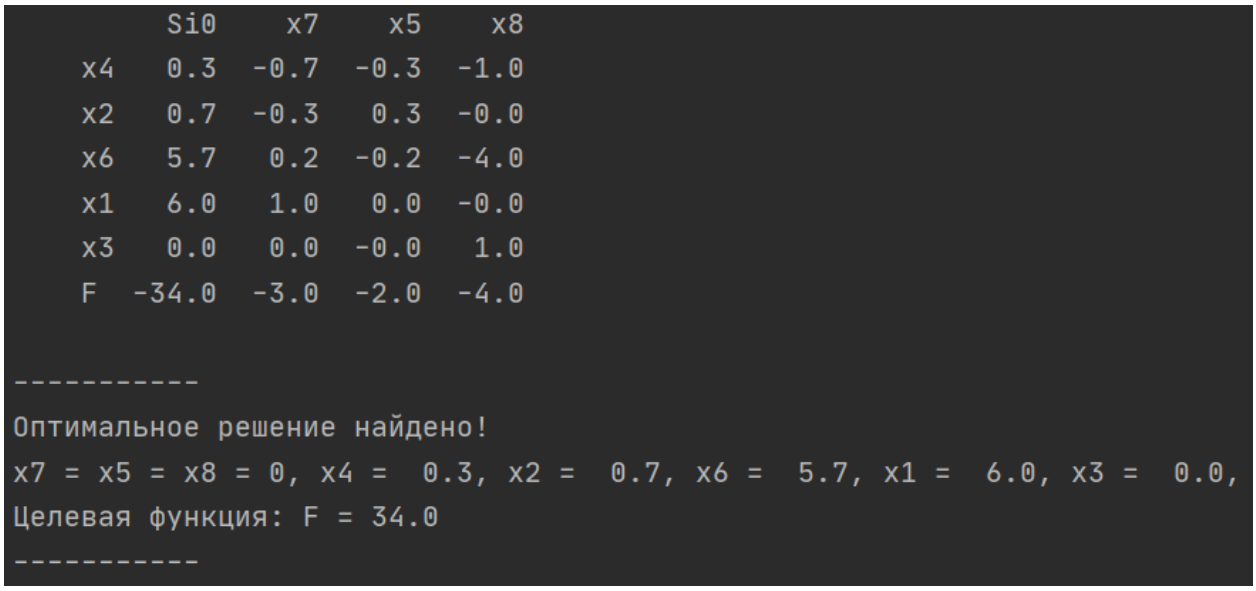


```
Исходная симплекс-таблица:
        Si0    x1    x2    x3
    x4   7.0   1.0   1.0   1.0
    x5   8.0   1.0   3.0   0.0
    x6   6.0   0.0   0.5   4.0
    x7   6.0   1.0   0.0   0.0
    x8   0.0   0.0   0.0   1.0
    F    0.0   5.0   6.0   4.0

-----------
Опорное решение найдено!
x1 = x2 = x3 = 0, x4 =  7.0, x5 =  8.0, x6 =  6.0, x7 =  6.0, x8 =  0.0,
Целевая функция: F = 0.0
-----------
```

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 5 Конечная симплекс-таблица.



```
        Si0    x7    x5    x8
    x4   0.3  -0.7  -0.3  -1.0
    x2   0.7  -0.3   0.3  -0.0
    x6   5.7   0.2  -0.2  -4.0
    x1   6.0   1.0   0.0  -0.0
    x3   0.0   0.0  -0.0   1.0
    F  -34.0  -3.0  -2.0  -4.0

-----------
Оптимальное решение найдено!
x7 = x5 = x8 = 0, x4 =  0.3, x2 =  0.7, x6 =  5.7, x1 =  6.0, x3 =  0.0,
Целевая функция: F = 34.0
-----------
```

Полученное решение $x_1 = 6,\ x_2 = 0.7,\ x_3 = 0; F = 34.$ целочисленным не является.

Осуществим ветвление по переменной x_2.

1.1.1)Введём новое ограничение $x_2 \leq 0$ и получим задачу ЦЛП:

$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 + x_7 = 6 \\ x_3 + x_8 = 0 \\ x_2 + x_9 = 0 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9 \in Z_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 6 Исходная симплекс-таблица.

```
Исходная симплекс-таблица:
         Si0    x1    x2    x3
    x4   7.0   1.0   1.0   1.0
    x5   8.0   1.0   3.0   0.0
    x6   6.0   0.0   0.5   4.0
    x7   6.0   1.0   0.0   0.0
    x8   0.0   0.0   0.0   1.0
    x9   0.0   0.0   1.0   0.0
    F    0.0   5.0   6.0   4.0

-----------
Опорное решение найдено!
x1 = x2 = x3 = 0, x4 =  7.0, x5 =  8.0, x6 =  6.0, x7 =  6.0, x8 =  0.0, x9 =  0.0,
Целевая функция: F = 0.0
-----------
```

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 7 Конечная симплекс-таблица.

```
         Si0    x7    x9    x8
    x4   1.0  -1.0  -1.0  -1.0
    x5   2.0  -1.0  -3.0  -0.0
    x6   6.0   0.0  -0.5  -4.0
    x1   6.0   1.0   0.0  -0.0
    x3   0.0   0.0  -0.0   1.0
    x2   0.0  -0.0   1.0  -0.0
    F  -30.0  -5.0  -6.0  -4.0

-----------
Оптимальное решение найдено!
x7 = x9 = x8 = 0, x4 =  1.0, x5 =  2.0, x6 =  6.0, x1 =  6.0, x3 =  0.0, x2 =  0.0,
Целевая функция: F = 30.0
-----------
```

Получаем целочисленное решение:

$$x_1 = 6,\ x_2 = 0,\ x_3 = 0;\ F = 30.$$

1.1.2)Введём новое ограничение $x_2 \geq 1$ и получим задачу ЦЛП:

$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 + x_7 = 6 \\ x_3 + x_8 = 0 \\ x_2 - x_9 = 1 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9 \in Z_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 8 Исходная симплекс-таблица.

Исходная симплекс-таблица:

	Si0	x1	x2	x3
x4	7.0	1.0	1.0	1.0
x5	8.0	1.0	3.0	0.0
x6	6.0	0.0	0.5	4.0
x7	6.0	1.0	0.0	0.0
x8	0.0	0.0	0.0	1.0
x9	-1.0	0.0	-1.0	0.0
F	0.0	5.0	6.0	4.0

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 9 Конечная симплекс-таблица.

	Si0	x5	x9	x8
x4	1.0	-1.0	-2.0	-1.0
x1	5.0	1.0	3.0	-0.0
x6	5.5	0.0	0.5	-4.0
x7	1.0	-1.0	-3.0	-0.0
x3	0.0	-0.0	0.0	1.0
x2	1.0	0.0	-1.0	-0.0
F	-31.0	-5.0	-9.0	-4.0

```
-----------
Оптимальное решение найдено!
x5 = x9 = x8 = 0, x4 =  1.0, x1 =  5.0, x6 =  5.5, x7 =  1.0, x3 =  0.0, x2 =  1.0,
Целевая функция: F = 31.0
-----------
```

Получаем целочисленное решение:

$$x_1 = 5,\ x_2 = 1,\ x_3 = 0;\ F = 31.$$

1.2)Введём новое ограничение $x_3 \geq 1$ и получим задачу ЦЛП:

$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 + x_7 = 6 \\ x_3 - x_8 = 1 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in Z_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 10 Исходная симплекс-таблица.

```
Исходная симплекс-таблица:
         Si0     x1     x2     x3
    x4   7.0    1.0    1.0    1.0
    x5   8.0    1.0    3.0    0.0
    x6   6.0    0.0    0.5    4.0
    x7   6.0    1.0    0.0    0.0
    x8  -1.0    0.0    0.0   -1.0
    F    0.0    5.0    6.0    4.0
```

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 11 Конечная симплекс-таблица.

```
         Si0    x4    x5    x8
    x1   5.0   1.5  -0.5   1.5
    x2   1.0  -0.5   0.5  -0.5
    x6   1.5   0.2  -0.2   4.2
    x7   1.0  -1.5   0.5  -1.5
    x3   1.0   0.0  -0.0  -1.0
    F  -35.0  -4.5  -0.5  -0.5

-----------
Оптимальное решение найдено!
x4 = x5 = x8 = 0, x1 =  5.0, x2 =  1.0, x6 =  1.5, x7 =  1.0, x3 =  1.0,
Целевая функция: F = 35.0
```

Получаем целочисленное решение:

$$x_1 = 5,\ x_2 = 1,\ x_3 = 1;\ F = 35.$$

1.2)Введём новое ограничение $x_1 \geq 7$ и получим задачу ЦЛП:

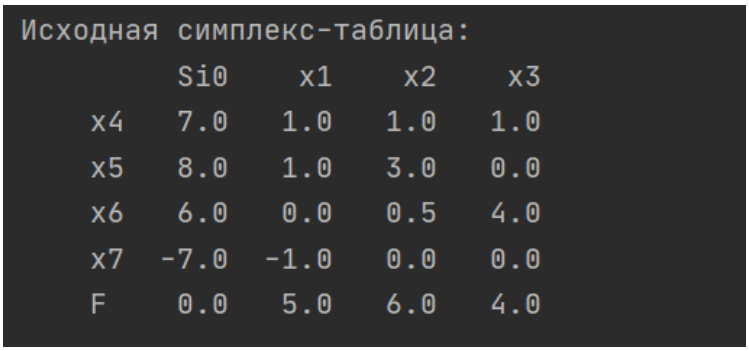
$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + 4x_3 + x_6 = 8 \\ x_1 - x_7 = 7 \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \in Z_+^0$$

Соответствующая симплекс-таблица имеет вид:

Рисунок 12 Исходная симплекс-таблица.

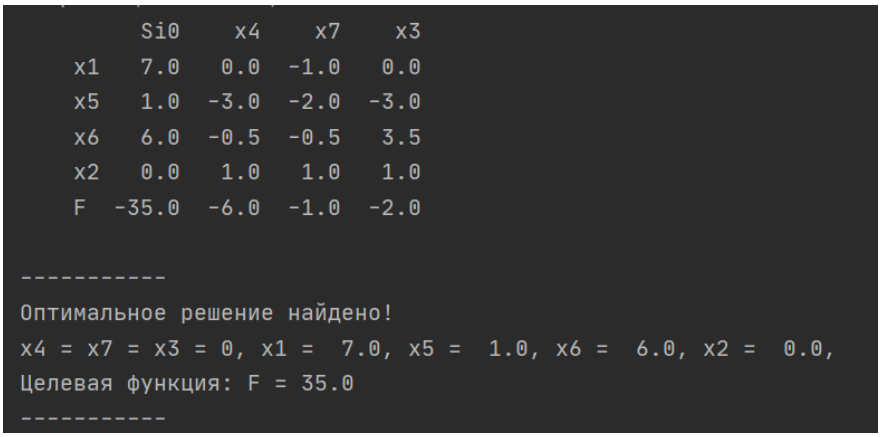


Исходная симплекс-таблица:

	Si0	x1	x2	x3
x4	7.0	1.0	1.0	1.0
x5	8.0	1.0	3.0	0.0
x6	6.0	0.0	0.5	4.0
x7	-7.0	-1.0	0.0	0.0
F	0.0	5.0	6.0	4.0

После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 13 Конечная симплекс-таблица.



	Si0	x4	x7	x3
x1	7.0	0.0	-1.0	0.0
x5	1.0	-3.0	-2.0	-3.0
x6	6.0	-0.5	-0.5	3.5
x2	0.0	1.0	1.0	1.0
F	-35.0	-6.0	-1.0	-2.0

Оптимальное решение найдено!

x4 = x7 = x3 = 0, x1 = 7.0, x5 = 1.0, x6 = 6.0, x2 = 0.0,

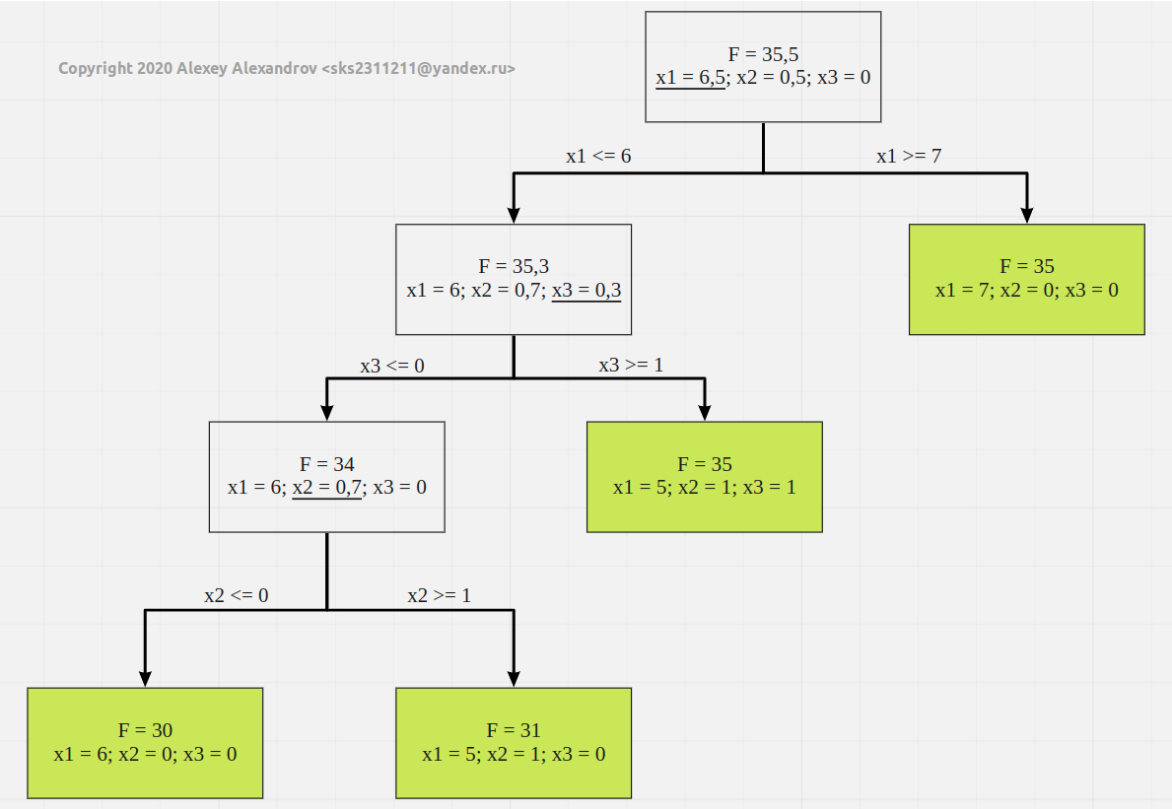
Целевая функция: F = 35.0

Получаем целочисленное решение:

$$x_1 = 7,\ x_2 = 0,\ x_3 = 0;\ F = 35.$$

В итоге мы получили такое дерево решений:

Рисунок 14 Дерево решений задачи ЦЛП



Оптимальное решение:

Рисунок 15 Оптимальное решение задачи ЦЛП.

```
-----------
В процессе решения задачи ЦЛП методом ветвей и границ найдены решения:
F = 30.0; x1 = 6.0, x2 = 0.0, x3 = 0.0
F = 31.0; x1 = 5.0, x2 = 1.0, x3 = 0.0
F = 35.0; x1 = 5.0, x2 = 1.0, x3 = 1.0
F = 35.0; x1 = 7.0, x2 = 0.0, x3 = 0
------------------
Тогда решением задачи ЦЛП будет являться решение:
F = 35.0; x1 = 5.0, x2 = 1.0, x3 = 1.0
 ------------------


Process finished with exit code 0
```

$$x_1 = 5,\ x_2 = 1,\ x_3 = 1;\ F = 35.$$

Проверка полученного решения:

$$F(5,1,1) = 5 \cdot 5 + 6 \cdot 1 + 4 \cdot 1 = 35;$$

$$\begin{cases} 5 + 1 + 1 \leq 7 \\ 5 + 3 \cdot 1 \leq 8 \\ 0.5 \cdot 1 + 4 \cdot 1 \leq 8 \end{cases};$$

$$5, 1, 1 \in Z_+^0$$

Вывод:

В ходе проделанной работы я изучил постановку задачи целочисленного линейного программирования, получил навыки решения задачи ЦЛП методом ветвей и границ. Данный метод будет полезен для решения других задач.

МВГ является улучшенной версией метода полного перебора, так как на каждом шаге ветвления мы фактически отсекаем большинство ненужных нам значений переменных, и скорость решения задачи ЦЛП увеличивается.

Приложение А

Код программы:

Файл “main.py”

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

"""
Лабораторная работа № 3
Целочисленное линейное программирование. Метод ветвей и границ.
Цель работы: изучить постановку задачи ЦЛП; получить навыки
решения задачи ЦЛП методом ветвей и границ.

Вариант 1.
"""

from simplex import *
from branch_and_bound import *
import brute_force_method
import branch_and_bound

if __name__ == '__main__':
    problem = Simplex("input_data.json")

    print(problem)
    # Находим опорное решение задачи ЛП
    problem.reference_solution()
    # Находим оптимальное решение задачи ЛП
    problem.optimal_solution()

    # Находим рещение задачи ЦЛП полным перебором.
    brute_force = brute_force_method.BruteForceMethod("input_data.json")
    print(brute_force)

    # Переходим к методу ветвей и границ.
    bb = branch_and_bound.BranchAndBound(problem)
    print(bb)
```

Файл “simplex.py”

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

from simplex_table import *
```

```
import json


class Simplex:
    """
    Класс для решения задачи ЛП симплекс-методом.
    """

    def __init__(self, path_to_file):
        """
        Переопределённый метод __init__. Регистрирует входные данные из JSON-файла.
        Определяем условие задачи.
        :param path_to_file: путь до JSON-файла с входными данными.
        """

        # Парсим JSON-файл с входными данными
        with open(path_to_file, "r") as read_file:
            json_data = json.load(read_file)
            self.obj_func_coffs_ = np.array(json_data["obj_func_coffs"])  # вектор-строка с - коэффициенты ЦФ
            self.constraint_system_lhs_ = np.array(json_data["constraint_system_lhs"])  # матрица ограничений A
            self.constraint_system_rhs_ = np.array(json_data["constraint_system_rhs"])  # вектор-столбец ограничений b
            self.func_direction_ = json_data["func_direction"]  # направление задачи (min или max)

            if len(self.constraint_system_rhs_) != self.constraint_system_rhs_.shape[0]:
                raise SimplexException(
                    "Ошибка при вводе данных. Число строк в матрице и столбце ограничений не совпадает.")

            # Если задача на max, то меняем знаки ЦФ и направление задачи (в конце возьмем решение со знаком минус и
            # получим искомое).
            if self.func_direction_ == "max":
                self.obj_func_coffs_ *= -1

            # Инициализация симплекс-таблицы.
            self.simplex_table_ = SimplexTable(self.obj_func_coffs_, self.constraint_system_lhs_,
                                               self.constraint_system_rhs_)
```

```
    def __str__(self):
        """
        Переопренный метод __str__ для условия задачи.
        :return: Строка с выводом условия задачи.
        """
        output = """Условие задачи:
----------------------------------------------------------
Найти вектор x = (x1,x2,..., xn)^T как решение след. задачи:"""

        output += f"\nF = cx -> {self.func_direction_},"
        output += "\nAx <= b,\nx1,x2, ..., xn >= 0"
        output += f"\nC = {self.obj_func_coffs_},"
        output += f"\nA =\n{self.constraint_system_lhs_},"
        output += f"\nb^T = {self.constraint_system_rhs_}."
        output += "\n----------------------------------------------------------"

        return output

    # Этап 1. Поиск опорного решения.
    def reference_solution(self):
        """
        Метод производит отыскание опорного решения.
        """
        print("Процесс решения:\n1) Поиск опорного решения:")
        print("Исходная симплекс-таблица:", self.simplex_table_, sep="\n")

        while not self.simplex_table_.is_find_ref_solution():
            self.simplex_table_.search_ref_solution()

        print("-----------")
        print("Опорное решение найдено!")
        self.output_solution()
        print("-----------")

    # Этап 2. Поиск оптимального решения.
    def optimal_solution(self):
        """
        Метод производит отыскание оптимального решения.
        """
        print("2) Поиск оптимального решения:")

        while not self.simplex_table_.is_find_opt_solution():
            self.simplex_table_.optimize_ref_solution()

        # Если задача на max, то в начале свели задачу к поиску min, а теперь
```

```
        # возьмём это решение со знаком минус и получим ответ для мак.
        if self.func_direction_ == "max":
            self.simplex_table_.main_table_[self.simplex_table_.main_table_.shape[0]
- 1][0] *= -1

        print("-----------")
        print("Оптимальное решение найдено!")
        self.output_solution()
        print("-----------")

    def output_solution(self):
        """
        Метод выводит текущее решение, используется для вывода опорного и
потимального решений.
        """
        fict_vars = self.simplex_table_.top_row_[2:]
        last_row_ind = self.simplex_table_.main_table_.shape[0] - 1

        for var in fict_vars:
            print(var, "= ", end="")
        print(0, end=", ")

        for i in range(last_row_ind):
            print(self.simplex_table_.left_column_[i], "= ",
round(self.simplex_table_.main_table_[i][0], 1), end=", ")

        print("\nЦелевая функция: F =",
round(self.simplex_table_.main_table_[last_row_ind][0], 1))
```

Файл “simplex_table.py”

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

import numpy as np


class SimplexException(Exception):
    """Пользовательское исключение для решения задач симплекс-методом."""

    def __init__(self, text):
        self.txt = text


class SimplexTable:
```

```
    """
    Класс симплекс-таблицы.
    """

    def __init__(self, obj_func_coffs, constraint_system_lhs, con-
straint_system_rhs):
        """
        Переопределённый метод __init__ для создания экземпляра класса Sim-
plexTable.
        :param obj_func_coffs: коэффициенты ЦФ.
        :param constraint_system_lhs: левая часть системы ограничений.
        :param constraint_system_rhs: правая часть системы ограничений.
        """
        var_count = len(obj_func_coffs)
        constraint_count = constraint_system_lhs.shape[0]

        # Заполнение верхнего хедера.
        self.top_row_ = ["  ", "Si0"]
        for i in range(var_count):
            self.top_row_.append("x" + str(i + 1))

        # Заполнение левого хедера.
        self.left_column_ = []
        ind = var_count
        for i in range(constraint_count):
            ind += 1
            self.left_column_.append("x" + str(ind))
        self.left_column_.append("F ")

        self.main_table_ = np.zeros((constraint_count + 1, var_count + 1))
        # Заполняем столбец Si0.
        for i in range(constraint_count):
            self.main_table_[i][0] = constraint_system_rhs[i]
        # Заполняем строку F.
        for j in range(var_count):
            self.main_table_[constraint_count][j + 1] = -obj_func_coffs[j]

        # Заполняем A.
        for i in range(constraint_count):
            for j in range(var_count):
                self.main_table_[i][j + 1] = constraint_system_lhs[i][j]

    def __str__(self):
        """
        Переопренный метод __str__ для симплекс-таблицы.
```

```
        :return: Строка с выводом симплекс-таблицы.
        """
        output = ""
        for header in self.top_row_:
            output += '{:>6}'.format(header)
        output += "\n"

        for i in range(self.main_table_.shape[0]):
            output += '{:>6}'.format(self.left_column_[i])
            for j in range(self.main_table_.shape[1]):
                output += '{:>6}'.format(round(self.main_table_[i][j], 1))
            output += "\n"

        return output

    def is_find_ref_solution(self):
        """
        Функция проверяет, найдено ли опорное решение по свободным в сим-
плекс-таблице.
        :return: True - опорное решение уже найдено. False - полученное решение
пока не является опорным.
        """

        # Проверяем все, кроме коэффициента ЦФ
        for i in range(self.main_table_.shape[0] - 1):
            if self.main_table_[i][0] < 0:
                return False
        return True

    def search_ref_solution(self):
        """
        Функция производит одну итерацию поиска опорного решения.
        """
        res_row = None
        for i in range(self.main_table_.shape[0] - 1):
            if self.main_table_[i][0] < 0:
                res_row = i
                break

        # Если найден отрицательный элемент в столбце свободных членов, то
ищем первый отрицательный в строке с ней.
        res_col = None
        if res_row is not None:
            for j in range(1, self.main_table_.shape[1]):
                if self.main_table_[res_row, j] < 0:
```

```
                res_col = j
                break

        # Если найден разрешающий столбец, то находим в нём разрешающий элемент.
        res_element = None
        if res_col is not None:
            # Ищем минимальное положительное отношение Si0 / x[res_col]
            minimum = None
            ind = -1
            # for i in range(self.main_table_.shape[0] - 1):
            #     curr = self.main_table_[i][0] / self.main_table_[i][res_col]
            #     if curr <= 0 or (self.main_table_[i][res_col] == 0):
            #         continue
            #     elif minimum is None:
            #         minimum = curr
            #         ind = i
            #     elif curr < minimum:
            #         minimum = curr
            #         ind = i
            for i in range(self.main_table_.shape[0] - 1):
                # Ищем минимальное отношение -- разрешающую строку.
                curr = self.main_table_[i][res_col]
                s_i0 = self.main_table_[i][0]
                if curr == 0:
                    continue
                elif (s_i0 / curr) > 0 and (minimum is None or (s_i0 / curr) < minimum):
                    minimum = (s_i0 / curr)
                    ind = i

            if minimum is None:
                raise SimplexException("Решения не существует! При нахождении опорного решения не нашлось минимального "
                                       "положительного отношения.")
            else:
                res_row = ind

            # Разрешающий элемент найден.
            res_element = self.main_table_[res_row][res_col]
            print("Разрешающая строка: {}".format(self.left_column_[res_row]))
            print("Разрешающий столбец: {}".format(self.top_row_[res_col + 1]))

            # Пересчёт симплекс-таблицы.
            self.recalc_table(res_row, res_col, res_element)
        else:
```

```
            raise SimplexException("Задача не имеет допустимых решений! При нахождении опорного решения не нашлось "
                                   "отрицательного элемента в строке с отрицательным свободным членом.")

    def is_find_opt_solution(self):
        """
        Функция проверяет, найдено ли оптимальное решение по коэффициентам ЦФ в симплекс-таблице.
        :return: True - оптимальное решение уже найдено. False - полученное решение пока не оптимально.
        """

        for i in range(1, self.main_table_.shape[1]):
            if self.main_table_[self.main_table_.shape[0] - 1][i] > 0:
                return False
        # Если положительных не нашлось, то оптимальное решение уже найдено.
        return True

    def optimize_ref_solution(self):
        """
        Функция производит одну итерацию поиска оптимального решения на основе
        уже полученного опорного решения.
        """
        res_col = None
        ind_f = self.main_table_.shape[0] - 1

        # В строке F ищем первый положительный.
        for j in range(1, self.main_table_.shape[1]):
            curr = self.main_table_[ind_f][j]
            if curr > 0:
                res_col = j
                break

        minimum = None
        res_row = None

        # Идём по всем, кроме ЦФ ищём минимальное отношение.
        for i in range(self.main_table_.shape[0] - 1):
            # Ищем минимальное отношение -- разрешающую строку.
            curr = self.main_table_[i][res_col]
            s_i0 = self.main_table_[i][0]
            if curr < 0:
```

```
                continue
            elif (s_i0 / curr) >= 0 and (minimum is None or (s_i0 / curr) < minimum):
                minimum = (s_i0 / curr)
                res_row = i

        if res_row is None:
            raise SimplexException("Функция не ограничена! Оптимального решения не существует.")

        else:
            # Разрешающий элемент найден.
            res_element = self.main_table_[res_row][res_col]
            print("Разрешающая строка: {}".format(self.left_column_[res_row]))
            print("Разрешающий столбец: {}".format(self.top_row_[res_col + 1]))
            # Пересчёт симплекс-таблицы.
            self.recalc_table(res_row, res_col, res_element)

    def recalc_table(self, res_row, res_col, res_element):
        """
        Функция по заданным разрешающим строке, столбцу и элекменту производит перерасчёт
        симплекс-таблицы методом жордановых искоючений.
        :param res_row: индекс разрешающей строки
        :param res_col: индекс разрешающего столбца
        :param res_element: разрешающий элемент
        """

        recalced_table = np.zeros((self.main_table_.shape[0], self.main_table_.shape[1]))

        # Пересчёт разрешающего элемента.
        recalced_table[res_row][res_col] = 1 / res_element

        # Пересчёт разрешающей строки.
        for j in range(self.main_table_.shape[1]):
            if j != res_col:
                recalced_table[res_row][j] = self.main_table_[res_row][j] / res_element

        # Пересчёт разрешающего столбца.
        for i in range(self.main_table_.shape[0]):
            if i != res_row:
                recalced_table[i][res_col] = -(self.main_table_[i][res_col] / res_element)

        # Пересчёт оставшейся части таблицы.
        for i in range(self.main_table_.shape[0]):
```

```
            for j in range(self.main_table_.shape[1]):
                if (i != res_row) and (j != res_col):
                    recalced_table[i][j] = self.main_table_[i][j] - (
                            (self.main_table_[i][res_col] * self.main_table_[res_row][j]) /
res_element)

        self.main_table_ = recalced_table

        self.swap_headers(res_row, res_col)

        print(self.__str__())

    def swap_headers(self, res_row, res_col):
        """
        Функция меняет меняет переменные в строке и столбце местами.
        :param res_row: разрешающая строка
        :param res_col: разрешающий столбец
        """

        temp = self.top_row_[res_col + 1]
        self.top_row_[res_col + 1] = self.left_column_[res_row]
        self.left_column_[res_row] = temp
```

Файл “branch_and_bound.py”

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

import math

from simplex_b_b import *


class BranchAndBound:
    """Класс задачи ветвей и границ."""

    def __init__(self, simplex_problem):
        """
        Переопределённый метод __init__. Определяем условие задачи на основе
задачи ЛП.
        :param simplex_problem: задача ЛП.
        """

        self.obj_func_coffs_ = simplex_problem.obj_func_coffs_  # вектор-строка
с - коэффициенты ЦФ.
        self.constraint_system_lhs_ = simplex_problem.constraint_system_lhs_  #
```

```
        матрица ограничений A.
        self.constraint_system_rhs_ = simplex_problem.constraint_system_rhs_  # вектор-столбец ограничений b.
        self.func_direction_ = simplex_problem.func_direction_  # направление задачи (min или max).
        self.simplex_table_ = simplex_problem.simplex_table_  # симплекс-таблица

        self.extra_constraints_lhs_ = []  # дополнительные строки ограничений для матрицы A.
        self.extra_constraints_rhs_ = []  # дополнительные элементы ограничений для столбца b.
        self.solutions_storage_ = []  # хранилище целочисленных решений, найденных в ходе ветвления.

        if not is_integer_solution(self.simplex_table_):
            self.branching(self.simplex_table_, self.extra_constraints_lhs_, self.extra_constraints_rhs_)

    def __str__(self):
        """
        Переопренный метод __str__ для вывода задачи ЦЛП методом ветвей и границ.
        :return: Строка с выводом задачи.
        """

        output = "В процессе решения задачи ЦЛП методом ветвей и границ найдены решения:\n"
        for solution in self.solutions_storage_:
            output += f"F = {solution[0]}; x1 = {solution[1][0]}, x2 = {solution[1][1]}, x3 = {solution[1][2]}\n"

        best_solution = self.find_best_solution()
        output += "--------------------\n"
        output += f"Тогда решением задачи ЦЛП будет являться решение:\n" \
                  f"F = {best_solution[0]}; " \
                  f"x1 = {best_solution[1][0]}, " \
                  f"x2 = {best_solution[1][1]}, " \
                  f"x3 = {best_solution[1][2]}\n "
        output += "--------------------"
        return output

    def branching(self, simplex_table, extra_constraints_lhs, extra_constraints_rhs):
        """
        Метод производит ветвление на основе текущей симплекс таблицы.
```

```
        :param simplex_table: текущая симплекс таблица.
        :param extra_constraints_lhs: дополнительные ограничения в левой части системы.
        :param extra_constraints_rhs: дополнительные ограничения в правой части системы.
        """

        # В столбце свободных членов симплекс таблицы ищем нецелую и нефиктивную переменную.
        b_var_number, b_var_value = branch_var_search(simplex_table)
        constraint_system_lhs = self.constraint_system_lhs_
        constraint_system_rhs = self.constraint_system_rhs_

        print(f"Осуществим ветвление по переменной x{b_var_number} = {round(b_var_value, 1)}")
        print(
            f"x{b_var_number} <= {math.floor(b_var_value)} и x{b_var_number} >= {math.ceil(b_var_value)} "
            f"(-x{b_var_number} <= {-math.ceil(b_var_value)})")

        # Строка ограничений для ветвления x <= [x*] (левая ветка)

        # Заготавливаем строку ограничений для левой части системы.
        lhs_constraint_temp_row = np.zeros(3)
        lhs_constraint_temp_row[b_var_number - 1] = 1

        extra_constraints_lhs.append(lhs_constraint_temp_row)
        extra_constraints_rhs.append(math.floor(b_var_value))

        for i in range(len(extra_constraints_rhs)):
            # Добавляем строку в левую часть системы ограничений.
            constraint_system_lhs = np.vstack((constraint_system_lhs, extra_constraints_lhs[i]))
            # Добавляем элемент в правую часть системы ограничений.
            constraint_system_rhs = np.append(constraint_system_rhs, extra_constraints_rhs[i])

        # Оборачиваем в try-except: если поднимется исключение в задаче, решение пойдёт дальше по другим веткам и узлам.
        try:
            left_branch_task = SimplexBB(self.obj_func_coffs_, constraint_system_lhs,
                                         constraint_system_rhs,
                                         self.func_direction_)
```

```
            print(f"Ограничение: x{b_var_number} <= {math.floor(b_var_value)}")
            left_branch_task.reference_solution()
            left_branch_task.optimal_solution()

            if not is_integer_solution(left_branch_task.simplex_table_):
                print("От левой ветки ветвимся дальше...")
                self.branching(left_branch_task.simplex_table_, extra_constraints_lhs,
extra_constraints_rhs)
            else:
                self.add_solution(left_branch_task.simplex_table_)

        except SimplexException as err:
            print(
                f"При ветвлении по x{b_var_number} <= {math.floor(b_var_value)}
Решение не было найдено:\n{err}")

        constraint_system_lhs = np.delete(constraint_system_lhs, con-
straint_system_lhs.shape[0] - 1, axis=0)
        constraint_system_rhs = np.delete(constraint_system_rhs, con-
straint_system_rhs.shape[0] - 1, axis=0)
        extra_constraints_lhs = extra_constraints_lhs[:-1]
        extra_constraints_rhs = extra_constraints_rhs[:-1]

        # Строка ограничений для ветвления x >= [x*] + 1 (правая ветка)

        # Заготавливаем строку ограничений для левой части системы.
        lhs_constraint_temp_row = np.zeros(3)
        lhs_constraint_temp_row[b_var_number - 1] = -1

        extra_constraints_lhs.append(lhs_constraint_temp_row)
        extra_constraints_rhs.append(-math.ceil(b_var_value))
        # Добавляем строку в левую часть системы ограничений.
        constraint_system_lhs = np.vstack((constraint_system_lhs,
lhs_constraint_temp_row))
        # Добавляем элемент в правую часть системы ограничений.
        constraint_system_rhs = np.append(constraint_system_rhs, -
math.ceil(b_var_value))

        # Оборачиваем в try-except: если поднимется исключение в задаче, реше-
ние пойдёт дальше по другим веткам и узлам.
        try:
            right_branch_task = SimplexBB(self.obj_func_coffs_, con-
straint_system_lhs,
                                          constraint_system_rhs,
                                          self.func_direction_)
```

```
            print(f"Ограничение: x{b_var_number} >= {math.ceil(b_var_value)}")
            right_branch_task.reference_solution()
            right_branch_task.optimal_solution()

            if not is_integer_solution(right_branch_task.simplex_table_):
                print("От правой ветки ветвимся дальше...")
                self.branching(right_branch_task.simplex_table_, extra_constraints_lhs,
extra_constraints_rhs)

            if is_integer_solution(right_branch_task.simplex_table_):
                self.add_solution(right_branch_task.simplex_table_)
                return

        except SimplexException as err:
            print(
                f"При ветвлении по x{b_var_number} >= {math.ceil(b_var_value)}
Решение не было найдено:\n{err}")

    def find_best_solution(self):
        """Метод производит отыскание наилучшего целочисленного решения
среди полученных в ходе ветвлений."""

        if self.func_direction_ == "max":
            maximum_solution = None
            for solution in self.solutions_storage_:
                if maximum_solution is None or maximum_solution[0] < solution[0]:
                    maximum_solution = solution

            return maximum_solution
        else:
            minimum_solution = None
            for solution in self.solutions_storage_:
                if minimum_solution is None or minimum_solution[0] > solution[0]:
                    minimum_solution = solution

            return minimum_solution

    def add_solution(self, simplex_table):
        """Метод добавляет целочисленное решение в self.solutions_storage_."""

        rows_num = simplex_table.main_table_.shape[0]

        f = simplex_table.main_table_[rows_num - 1][0]

        vars_values = [0, 0, 0]
```

```
        for i in range(rows_num - 1):
            if simplex_table.left_column_[i] in ["x1", "x2", "x3"]:
                vars_values[int(simplex_table.left_column_[i][1]) - 1] = sim-
plex_table.main_table_[i][0]

        self.solutions_storage_.append((f, vars_values))


def is_integer_solution(simplex_table):
    """
    Проверяет, является ли найденное решение целовчисленным.
    :param simplex_table: текущее состояние задачи ЦЛП.
    return: True - решение задачи ЛП целочисленно. False - не целочисленно.
    """

    for i in range(simplex_table.main_table_.shape[0]):
        if (simplex_table.left_column_[i] in ["x1", "x2", "x3"]) and (not sim-
plex_table.main_table_[i][0].is_integer()):
            return False

    return True


def branch_var_search(simplex_table):
    """
    Производит поиск переменной ветвления в симплекс-таблице.
    :param simplex_table: текущее состояние задачи ЦЛП.
    return: возвращает номер переменной ветвления и её значение в симплекс-
таблице.
    """

    for i in range(len(simplex_table.left_column_) - 1):
        if (not simplex_table.main_table_[i][0].is_integer()) and (sim-
plex_table.left_column_[i] in ["x1", "x2", "x3"]):
            return int(simplex_table.left_column_[i][1]), sim-
plex_table.main_table_[i][0]
```

Файл “brute_force_method.json”

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

import json
import numpy as np
```

```
class BruteForceMethod:
    """Класс для решения задачи ЦЛП путём полного перебора."""

    def __init__(self, path_to_file):
        """
        Переопределённый метод __init__. Регистрирует входные данные из JSON-файла.
        Определяем условие задачи.
        :param path_to_file: путь до JSON-файла с входными данными.
        """

        # Парсим JSON-файл с входными данными
        with open(path_to_file, "r") as read_file:
            json_data = json.load(read_file)
            self.obj_func_coffs_ = np.array(json_data["obj_func_coffs"])  # вектор-строка с - коэффициенты ЦФ
            self.constraint_system_lhs_ = np.array(json_data["constraint_system_lhs"])  # матрица ограничений A
            self.constraint_system_rhs_ = np.array(json_data["constraint_system_rhs"])  # вектор-столбец ограничений b
            self.func_direction_ = json_data["func_direction"]  # направление задачи (min или max)

            if len(self.constraint_system_rhs_) != self.constraint_system_rhs_.shape[0]:
                raise Exception("Ошибка при вводе данных. Число строк в матрице и столбце ограничений не совпадает.")

            if len(self.constraint_system_rhs_) > len(self.obj_func_coffs_):
                raise Exception("СЛАУ несовместна! Число уравнений больше числа переменных.")

            self.all_solutions_ = self.list_of_solutions()
            self.max_ind_, self.max_func_value_ = self.search_optimal_solution()

    def __str__(self):
        """
        Переопренный метод __str__ для вывода всех целочисленных решений, включая наилучшее.
        :return: Строка с выводом всех решений и наилучшего из них.
        """

        output = "Метод полного перебора.\n"
```

```
        output += f"Всего имеется {len(self.all_solutions_)} допустимых целочисленных решений:\n"
        for solution in self.all_solutions_:
            output += f"({solution[0]}, {solution[1]}, {solution[2]});\n"

        output += "-----------\n"
        output += "Полный перебор всех решений даёт решение:\n"
        output += f"x1 = {self.all_solutions_[self.max_ind_][0]}," \
                  f" x2 = {self.all_solutions_[self.max_ind_][1]}," \
                  f" x3 = {self.all_solutions_[self.max_ind_][2]}," \
                  f" F = {self.max_func_value_} \n"
        output += "-----------\n"

        return output

    def is_satisfies_constraints(self, x1, x2, x3):
        """
        Метод проверяет, удовлетворяет ли заданный набор значений переменных ограничениям.
        :return: True -- удовлетворяет ограничениям, False -- не удовлетворяет им.
        """
        for i in range(len(self.constraint_system_rhs_)):
            if self.constraint_system_lhs_[i][0] * x1 + \
                    self.constraint_system_lhs_[i][1] * x2 + \
                    self.constraint_system_lhs_[i][2] * x3 > self.constraint_system_rhs_[i]:
                return False

        return True

    def list_of_solutions(self):
        """
        Перебирает все возможные целочисленные значения переменных и добавляет их в список.
        :return: Список решений задачи ЦЛП.
        """
        maximum = 0
        list_of_solutions = []

        # Находим максимальное в столбце ограничений
        for i in range(len(self.constraint_system_lhs_)):
            if self.constraint_system_rhs_[i] > maximum:
                maximum = self.constraint_system_rhs_[i]
```

```
        for x1 in range(maximum + 1):
            for x2 in range(maximum + 1):
                for x3 in range(maximum + 1):
                    if self.is_satisfies_constraints(x1, x2, x3):
                        list_of_solutions.append(np.array([x1, x2, x3]))

        return list_of_solutions

    def search_optimal_solution(self):
        """
        В списке всех целочисленных решений производит отыскание максимального.
        :return: возвращает индекс решения задачи ЦЛП в списке целочисленных решений и сам набор значений переменных.
        """
        list_of_func_values = []

        # Получаем список
        for i in range(len(self.all_solutions_)):
            f = 0
            for j in range(len(self.obj_func_coffs_)):
                f += self.obj_func_coffs_[j] * self.all_solutions_[i][j]
            list_of_func_values.append(f)

        # В этом списке ищем максимальное значение ЦФ и находим её индекс.
        max_f_ind = 0
        for i in range(len(list_of_func_values)):
            if list_of_func_values[i] > list_of_func_values[max_f_ind]:
                max_f_ind = i

        return max_f_ind, list_of_func_values[max_f_ind]
```

Файл “input_data.json”

```
{
    "obj_func_coffs": [5, 6, 4],
    "constraint_system_lhs": [
        [1, 1, 1],
        [1, 3, 0],
        [0, 0.5, 4]
    ],
    "constraint_system_rhs": [7, 8, 6],
    "func_direction": "max"
}
```