**Прізвище:** Пасічник

**Ім’я:** Василь

**Група:** КН-405

**Кафедра.:** Кафедра Систем

Автоматизованого Проектування

**Дисципліна:** Теорія прийняття рішень

**Перевірив:** Кривий Р.З.

**Звіт**

До лабораторної роботи №5

На тему “Матричні ігри”

**Мета роботи:** Визначити основні поняття теорії ігор, властивості змішаних стратегій. Вивчити метод вирішення матричних ігор у змішаних стратегіях за допомогою введення до подвійних завдань лінійного програмування.

**Короткі теоретичні відомості**

У грі беруть участь два гравці: A і B. У розпорядженні кожного гравця є кінцеве безліч варіантів вибору - стратегій. Нехай - безліч стратегій гравця A, - безліч стратегій гравця B. З кожною парою стратегій пов'язаний платіж, який один з гравців виплачує іншому. Тобто, коли гравець А вибирає стратегію (свою i-ю стратегію), а гравець В - стратегію, то результатом такого вибору стає платіж. Оскільки стратегій кінцеве число, то платежі утворюють матрицю розмірності n x m, звану матрицею платежів (або матрицею гри). Рядки цієї матриці відповідають стратегіям гравця А, а стовпці - стратегіям гравця В.

**Порядок вирішення завдання:**

1) Вихідні дані беруть із варіантів індивідуальних завдань.

2) При вирішенні матричної гри потрібно вийти на наступні етапи:

1. Знайти сідлову точку і перевірити, чи має гра вирішення в чистих стратегій.

2. У випадку відсутності чистої стратегії, знайти рішення в оптимальних змішаних стратегіях

3. Спростити платіжну матрицю (перевірити матрицю на домінуючі рядки і стовбці).

4. Визначити оптимальні плани за допомогою одного з методів лінійного програмування.

5. Знайдіть рішення гри.

**Індивідуальне завдання (варіант 13):**

11;7;10;8;12

14;7;11;9;15

11;7;11;15;10

14;11;9;12;11

15;10;7;8;10

**Виконання:**

Дані для матричної гри зчитуються з зовнішнього файлу (Рис. 1)

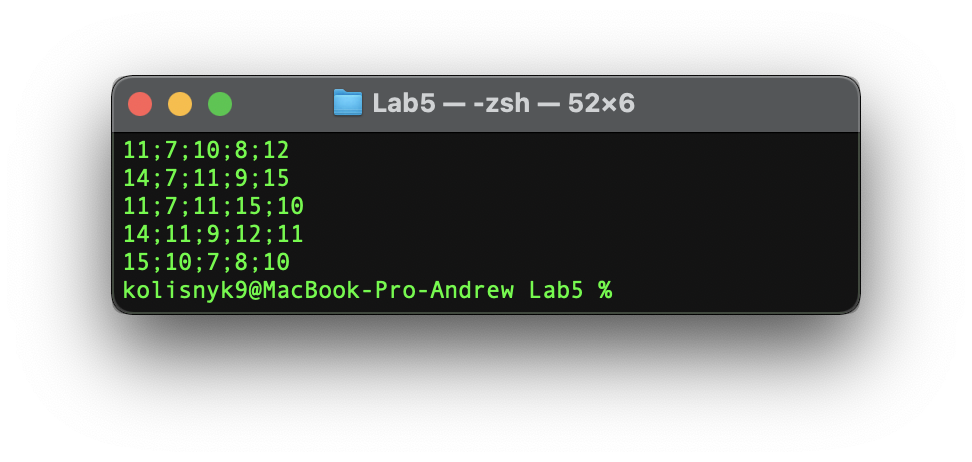


Рис. 1 Файл text.txt

Після отримання даних першим кроком є перевірка існування сідлової точки в платіжній матриці (рис. 1).

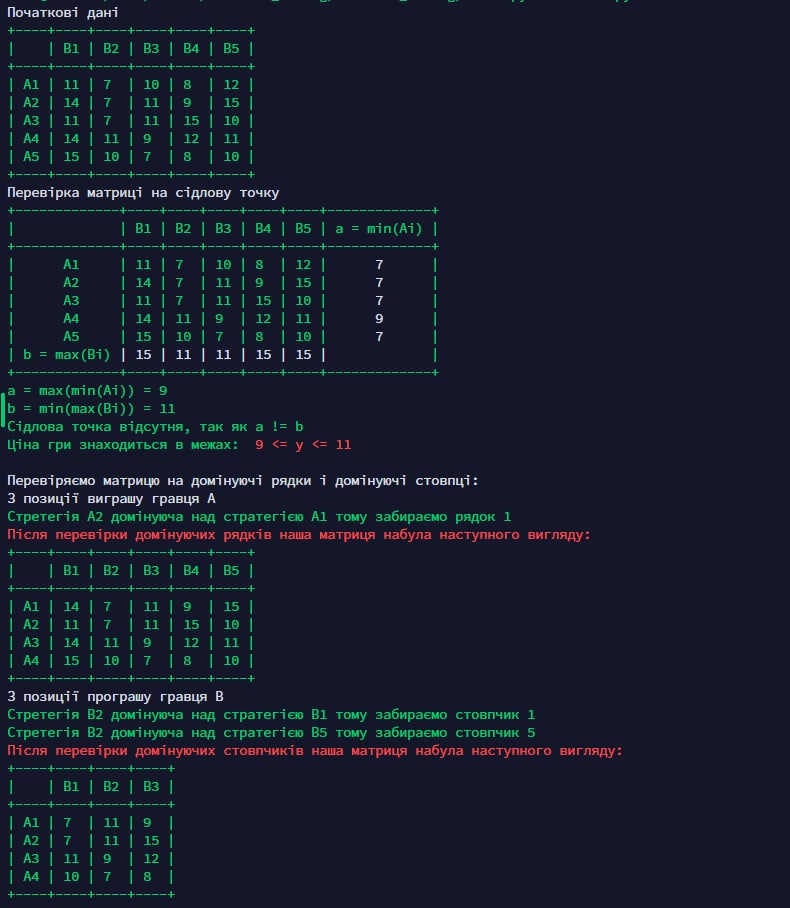


Рис. 1 Перевірка сідлової точки

Так як сідлова точка відсутня, то перевіряємо матрицю на домінуючі стовпці і домінуючі рядки (рис. 2)

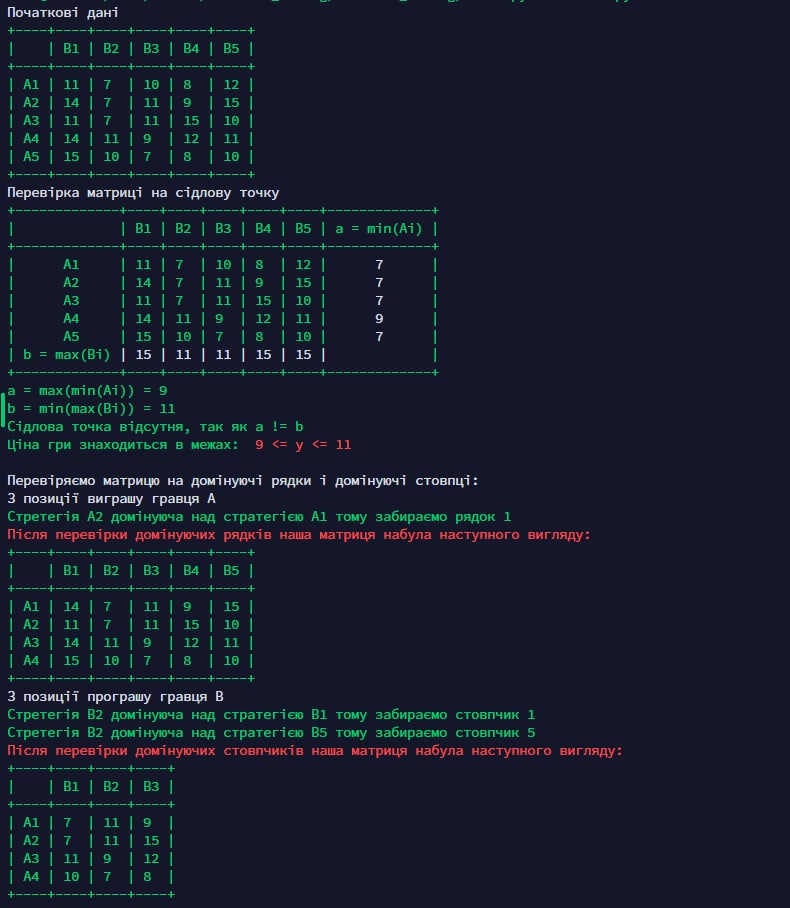


Рис. 2 Перевірка матриці на домінуючі стовпці і домінуючі рядки

Далі отриману матрицю записуємо у вигляді задачі лінійного програмування (рис. 3)

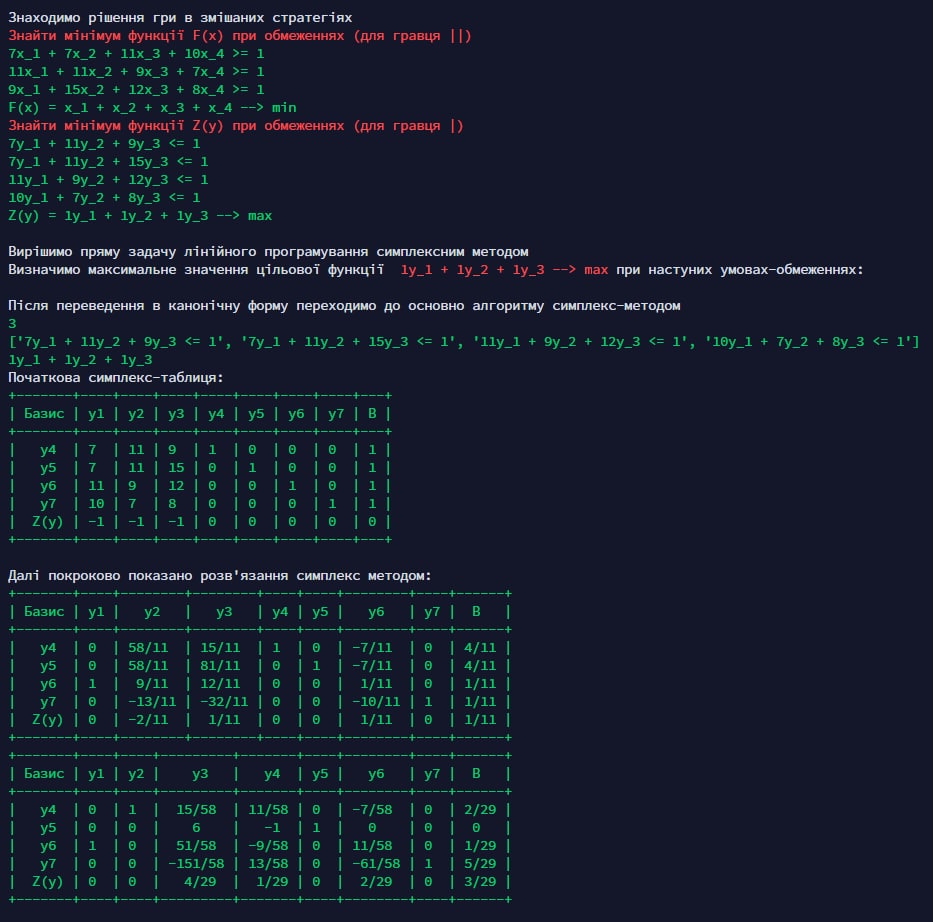


Рис. 3 Запис матриці у вигляді задачі лінійного програмування

Для розв’язання задачі лінійного програмування буде використовуватися симплекс метод. На рис. 4 зображено початкову симплекс таблицю.

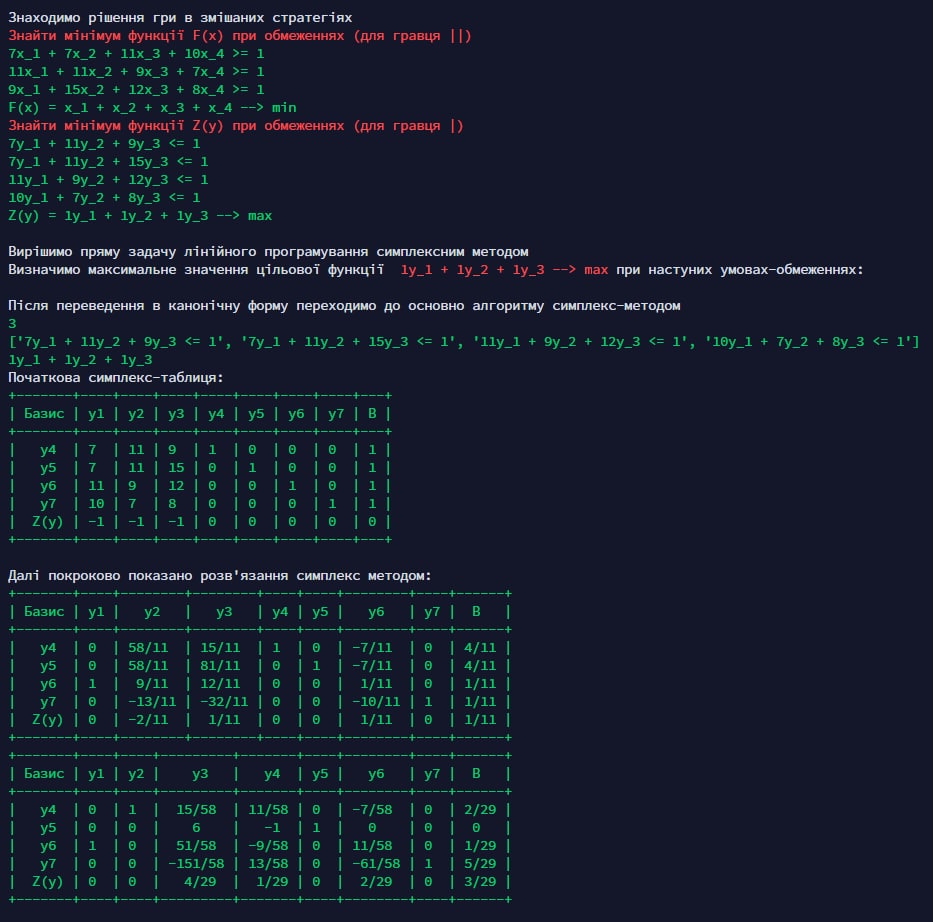
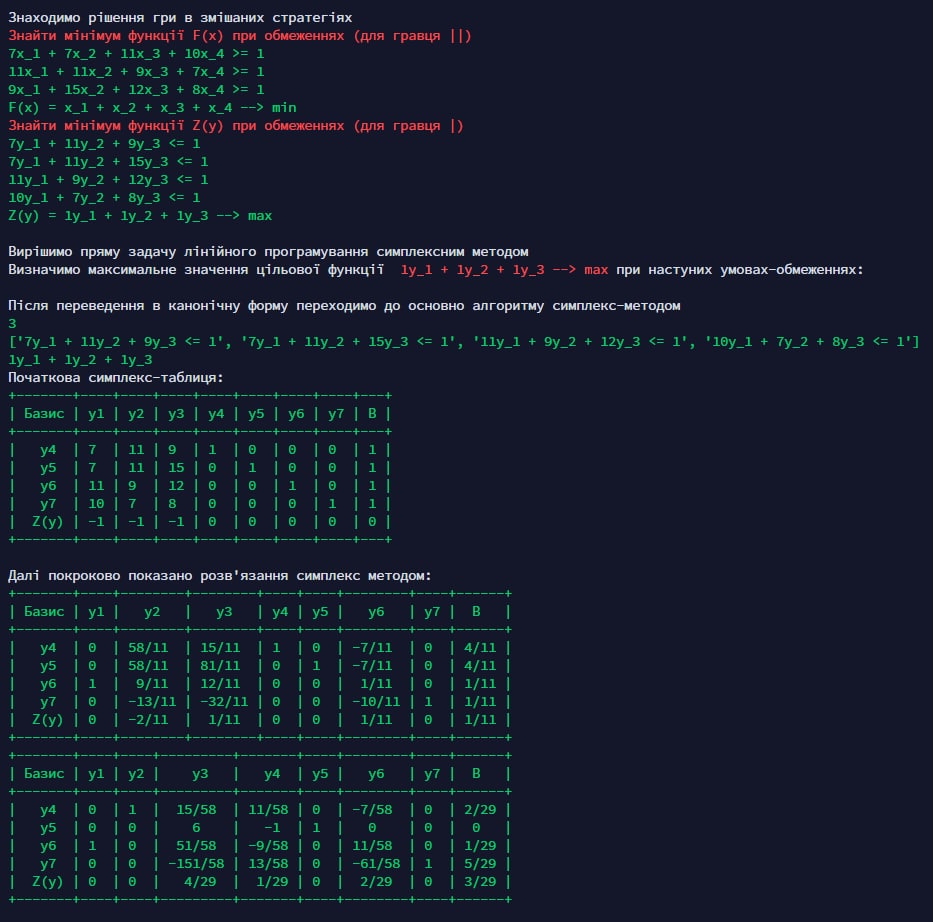


Рис. 4 Початкова симплекс таблиця

На рис. 5 зображено основний алгоритм симплекс методу.



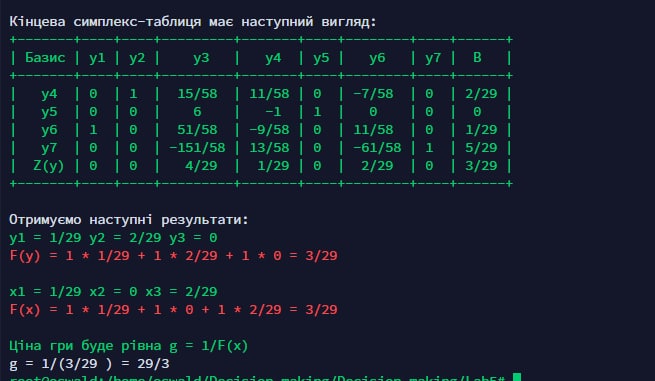


Рис. 5 Основний алгоритм симплекс методу

Результат, який вийшов після розв’язання задачі лінійного програмування можна побачити на рис. 6

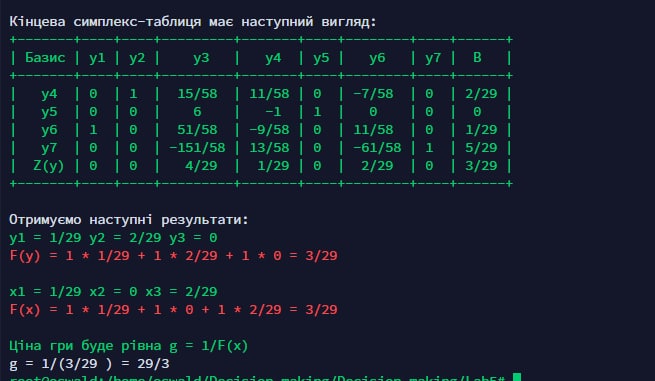


Рис. 6 Результати розв’язання

З отриманих результатів можемо зробити висновок, що ціна гри рівна 29/3

Повний код програми знаходиться за посиланням:

https://github.com/Kolisnyk9/Lab

**Код програми:**

**Файл main.py**

import sys

import re

import string

from copy import deepcopy

from colorama import Fore, Style, Back

from prettytable import PrettyTable

from simplex import Simplex

def open\_file():

if (len(sys.argv) < 2):

print ('Enter file name')

exit()

fileName = sys.argv[1]

try:

return open(fileName)

except FileNotFoundError:

print("Oops! File not exist...")

exit()

def get\_matrix\_table(matrix):

if len(matrix) < 1: return ''

x = PrettyTable()

fields = ['']

for i in range(len(matrix[0])):

fields.append("B" + str(i + 1))

x.field\_names = fields

for i in range(len(matrix)):

x.add\_row(['A' + str(i + 1)] + matrix[i])

return x

def check\_saddle\_point(minA, maxB):

maxFromMatrixA = minA[max(minA, key = minA.get)]

minFromMatrixB = maxB[min(maxB, key = maxB.get)]

return [maxFromMatrixA, minFromMatrixB]

def check\_rows(firstRow, secondRow):

equalElements = 0

for i in range(len(firstRow)):

if firstRow[i] < secondRow[i]: return 0

if firstRow[i] == secondRow[i]: equalElements += equalElements

return 0 if equalElements == len(firstRow) else 1

def check\_dominant\_rows(matrix):

matrixAfterExcludingRows = []

deletedRows = []

for i in range(len(matrix)):

for j in range(len(matrix)):

if i == j: continue

result = check\_rows(matrix[i], matrix[j])

if result != 0 and j not in deletedRows:

deletedRows.append(j)

print("Стретегія А" + str(i + 1), "домінуюча над стратегією А" + str(j + 1), "тому забираємо рядок", j + 1)

for i in range(len(matrix)):

if i in deletedRows: continue

matrixAfterExcludingRows.append(matrix[i])

return matrixAfterExcludingRows

def check\_dominant\_columns(matrix):

matrixAfterExcludingColumns = []

deletedColumns = []

transposedMatrix = [list(x) for x in zip(\*matrix)]

for i in range(len(transposedMatrix)):

for j in range(len(transposedMatrix)):

if i == j: continue

result = check\_rows(transposedMatrix[j], transposedMatrix[i])

if result != 0 and j not in deletedColumns:

deletedColumns.append(j)

print("Стретегія В" + str(i + 1), "домінуюча над стратегією В" + str(j + 1), "тому забираємо стовпчик", j + 1)

for i in range(len(matrix)):

matrixAfterExcludingColumns.append([])

for j in range(len(matrix[i])):

if j in deletedColumns: continue

matrixAfterExcludingColumns[i].append(matrix[i][j])

return matrixAfterExcludingColumns

file = open\_file()

matrix = []

for line in file:

matrix.append([int(d) for d in re.split(';', re.sub('\n', '', line))])

if len(matrix) < 2: exit()

print(Fore.WHITE + "Початкові дані", Style.RESET\_ALL)

print(get\_matrix\_table(matrix))

print(Fore.WHITE + "Перевірка матриці на сідлову точку", Style.RESET\_ALL)

x = PrettyTable()

fields = ['']

for i in range(len(matrix)):

fields.append("B" + str(i + 1))

fields.append('a = min(Ai)')

x.field\_names = fields

minA = {}

maxB = {}

for i in range(len(matrix)):

minA[i] = min(matrix[i])

x.add\_row(['A' + str(i + 1)] + matrix[i] + [Fore.WHITE + str(minA[i]) + Style.RESET\_ALL])

for j in range(len(matrix[i])):

if j not in maxB or maxB[j] < matrix[i][j]:

maxB[j] = matrix[i][j]

x.add\_row(['b = max(Bi)' + Fore.WHITE] + [maxB[element] for element in maxB] + ['' + Style.RESET\_ALL])

print(x)

[maxFromMatrixA, minFromMatrixB] = check\_saddle\_point(minA, maxB)

if maxFromMatrixA == minFromMatrixB:

print("Сідлова точка присутня!")

else:

print("a = max(min(Ai)) =", maxFromMatrixA)

print("b = min(max(Bi)) =", minFromMatrixB)

print("Сідлова точка відсутня, так як a != b")

print("Ціна гри знаходиться в межах:", Fore.RED, maxFromMatrixA, "<= y <=", minFromMatrixB, Style.RESET\_ALL)

print(Fore.WHITE, "\nПеревіряємо матрицю на домінуючі рядки і домінуючі стовпці:", Style.RESET\_ALL)

print(Fore.WHITE + "З позиції виграшу гравця А", Style.RESET\_ALL)

matrixAfterExcludingRows = check\_dominant\_rows(matrix)

print(Fore.RED + "Після перевірки домінуючих рядків наша матриця набула наступного вигляду: ", Style.RESET\_ALL)

print(get\_matrix\_table(matrixAfterExcludingRows))

print(Fore.WHITE + "З позиції програшу гравця В", Style.RESET\_ALL)

matrixAfterExcludingColumns = check\_dominant\_columns(matrixAfterExcludingRows)

print(Fore.RED + "Після перевірки домінуючих стовпчиків наша матриця набула наступного вигляду: ", Style.RESET\_ALL)

print(get\_matrix\_table(matrixAfterExcludingColumns))

transposedMatrix = [list(x) for x in zip(\*matrixAfterExcludingColumns)]

print(Fore.WHITE + "\nЗнаходимо рішення гри в змішаних стратегіях", Style.RESET\_ALL)

print(Fore.RED + "Знайти мінімум функції F(x) при обмеженнях (для гравця ||)", Style.RESET\_ALL)

secondPlayersConditions = []

for i in range(len(transposedMatrix)):

secondPlayersConditions.append('')

for j in range(len(transposedMatrix[i])):

secondPlayersConditions[i] += str(transposedMatrix[i][j]) + 'x\_' + str(j + 1) + ' + '

for i in range(len(secondPlayersConditions)):

print(secondPlayersConditions[i][:-2] + '>= 1')

mainCondition = 'F(x) = '

for i in range(len(matrixAfterExcludingColumns)):

mainCondition += 'x\_' + str(i + 1) + ' + '

print(mainCondition[:-2] + '--> min')

print(Fore.RED + "Знайти мінімум функції Z(y) при обмеженнях (для гравця |)", Style.RESET\_ALL)

firstPlayersConditions = []

vars\_count = 0

for i in range(len(matrixAfterExcludingColumns)):

firstPlayersConditions.append('')

columns = len(matrixAfterExcludingColumns[i])

for j in range(columns):

firstPlayersConditions[i] += str(matrixAfterExcludingColumns[i][j]) + 'y\_' + str(j + 1) + ' + '

if columns > vars\_count:

vars\_count = columns

conditions = ''

for i in range(len(firstPlayersConditions)):

firstPlayersConditions[i] = firstPlayersConditions[i][:-2] + '<= 1'

mainCondition = ''

for i in range(len(transposedMatrix)):

mainCondition += '1y\_' + str(i + 1) + ' + '

mainCondition = mainCondition[:-2]

for line in firstPlayersConditions:print(line)

print('Z(y) = ' + mainCondition + '--> max')

print(Fore.WHITE + "\nВирішимо пряму задачу лінійного програмування симплексним методом", Style.RESET\_ALL)

print(Fore.WHITE + "Визначимо максимальне значення цільової функції", Fore.RED, mainCondition + '--> max', Fore.WHITE + "при настуних умовах-обмеженнях:", Style.RESET\_ALL)

print(conditions)

print(Fore.WHITE + "Після переведення в канонічну форму переходимо до основно алгоритму симплекс-методом", Style.RESET\_ALL)

simplexResult = Simplex(num\_vars=vars\_count, constraints=firstPlayersConditions, objective\_function=mainCondition)

print(Fore.WHITE + "\nОтримуємо наступні результати:", Style.RESET\_ALL)

x\_result = {}

y\_result = {}

for key in simplexResult.solution:

if 'y\_' in key:

y\_result[key] = simplexResult.solution[key]

elif 'x\_' in key:

x\_result[key] = simplexResult.solution[key]

yResultCond = 'F(y) = '

yResult = 0

for i in range(vars\_count):

print('y' + str(i + 1) + ' =', y\_result['y\_' + str(i + 1)], end=' ')

yResult += 1 \* y\_result['y\_' + str(i + 1)]

yResultCond += "1 \* " + str(y\_result['y\_' + str(i + 1)]) + ' + '

print("\n" + Fore.RED + yResultCond[:-2] + '= ' + str(yResult), Style.RESET\_ALL, '\n')

xResultCond = 'F(x) = '

xResult = 0

for i in range(vars\_count):

print('x' + str(i + 1) + ' =', x\_result['x\_' + str(i + 1)], end=' ')

xResult += 1 \* x\_result['x\_' + str(i + 1)]

xResultCond += "1 \* " + str(x\_result['x\_' + str(i + 1)]) + ' + '

print("\n" + Fore.RED + xResultCond[:-2] + '= ' + str(xResult), Style.RESET\_ALL)

print("\nЦіна гри буде рівна g = 1/F(x)")

print(Fore.WHITE + "g = 1/(" + str(xResult), ") =", str(1/xResult), Style.RESET\_ALL)

**simple.py**

from fractions import Fraction

from warnings import warn

from prettytable import PrettyTable

from colorama import Fore, Style, Back

class Simplex(object):

def \_\_init\_\_(self, num\_vars, constraints, objective\_function):

self.num\_vars = num\_vars

self.constraints = constraints

self.objective\_function = objective\_function

self.coeff\_matrix, self.r\_rows, self.num\_s\_vars, self.num\_r\_vars = self.construct\_matrix\_from\_constraints()

del self.constraints

self.basic\_vars = [0 for i in range(len(self.coeff\_matrix))]

self.phase1()

r\_index = self.num\_r\_vars + self.num\_s\_vars

for i in self.basic\_vars:

if i > r\_index:

raise ValueError("Нездійсненне рішення")

self.delete\_r\_vars()

self.solution = self.objective\_maximize()

self.optimize\_val = self.coeff\_matrix[0][-1]

def print\_simplex\_table(self):

x = PrettyTable()

x.field\_names = ['Базис'] + ['y' + str(i + 1) for i in range(len(self.coeff\_matrix[0]) - 1)] + ['B']

for i in range(1, len(self.coeff\_matrix)):

x.add\_row(['y' + str(i + self.num\_vars)] + self.coeff\_matrix[i])

x.add\_row(['Z(y)'] + self.coeff\_matrix[0])

print(x)

def construct\_matrix\_from\_constraints(self):

num\_s\_vars = 0

num\_r\_vars = 0

for expression in self.constraints:

if '>=' in expression:

num\_s\_vars += 1

elif '<=' in expression:

num\_s\_vars += 1

num\_r\_vars += 1

elif '=' in expression:

num\_r\_vars += 1

total\_vars = self.num\_vars + num\_s\_vars + num\_r\_vars

coeff\_matrix = [[Fraction("0/1") for i in range(total\_vars+1)] for j in range(len(self.constraints)+1)]

s\_index = self.num\_vars

r\_index = self.num\_vars + num\_s\_vars

r\_rows = []

for i in range(1, len(self.constraints)+1):

constraint = self.constraints[i-1].split(' ')

for j in range(len(constraint)):

if '\_' in constraint[j]:

coeff, index = constraint[j].split('\_')

if constraint[j-1] == '-':

coeff\_matrix[i][int(index)-1] = Fraction("-" + coeff[:-1] + "/1")

else:

coeff\_matrix[i][int(index)-1] = Fraction(coeff[:-1] + "/1")

elif constraint[j] == '<=':

coeff\_matrix[i][s\_index] = Fraction("1/1")

s\_index += 1

elif constraint[j] == '>=':

coeff\_matrix[i][s\_index] = Fraction("-1/1")

coeff\_matrix[i][r\_index] = Fraction("1/1")

s\_index += 1

r\_index += 1

r\_rows.append(i)

elif constraint[j] == '=':

coeff\_matrix[i][r\_index] = Fraction("1/1")

r\_index += 1

r\_rows.append(i)

coeff\_matrix[i][-1] = Fraction(constraint[-1] + "/1")

return coeff\_matrix, r\_rows, num\_s\_vars, num\_r\_vars

def phase1(self):

r\_index = self.num\_vars + self.num\_s\_vars

for i in range(r\_index, len(self.coeff\_matrix[0])-1):

self.coeff\_matrix[0][i] = Fraction("-1/1")

coeff\_0 = 0

for i in self.r\_rows:

self.coeff\_matrix[0] = add\_row(self.coeff\_matrix[0], self.coeff\_matrix[i])

self.basic\_vars[i] = r\_index

r\_index += 1

s\_index = self.num\_vars

for i in range(1, len(self.basic\_vars)):

if self.basic\_vars[i] == 0:

self.basic\_vars[i] = s\_index

s\_index += 1

key\_column = max\_index(self.coeff\_matrix[0])

condition = self.coeff\_matrix[0][key\_column] > 0

while condition is True:

key\_row = self.find\_key\_row(key\_column = key\_column)

self.basic\_vars[key\_row] = key\_column

pivot = self.coeff\_matrix[key\_row][key\_column]

self.normalize\_to\_pivot(key\_row, pivot)

self.make\_key\_column\_zero(key\_column, key\_row)

key\_column = max\_index(self.coeff\_matrix[0])

condition = self.coeff\_matrix[0][key\_column] > 0

def find\_key\_row(self, key\_column):

min\_val = float("inf")

min\_i = 0

for i in range(1, len(self.coeff\_matrix)):

if self.coeff\_matrix[i][key\_column] > 0:

val = self.coeff\_matrix[i][-1] / self.coeff\_matrix[i][key\_column]

if val < min\_val:

min\_val = val

min\_i = i

if min\_val == float("inf"):

raise ValueError("Необмежене рішення")

if min\_val == 0:

warn("Error")

return min\_i

def normalize\_to\_pivot(self, key\_row, pivot):

for i in range(len(self.coeff\_matrix[0])):

self.coeff\_matrix[key\_row][i] /= pivot

def make\_key\_column\_zero(self, key\_column, key\_row):

num\_columns = len(self.coeff\_matrix[0])

for i in range(len(self.coeff\_matrix)):

if i != key\_row:

factor = self.coeff\_matrix[i][key\_column]

for j in range(num\_columns):

self.coeff\_matrix[i][j] -= self.coeff\_matrix[key\_row][j] \* factor

def delete\_r\_vars(self):

for i in range(len(self.coeff\_matrix)):

non\_r\_length = self.num\_vars + self.num\_s\_vars + 1

length = len(self.coeff\_matrix[i])

while length != non\_r\_length:

del self.coeff\_matrix[i][non\_r\_length-1]

length -= 1

def update\_objective\_function(self):

objective\_function\_coeffs = self.objective\_function.split()

for i in range(len(objective\_function\_coeffs)):

if '\_' in objective\_function\_coeffs[i]:

coeff, index = objective\_function\_coeffs[i].split('\_')

if objective\_function\_coeffs[i-1] == '-':

self.coeff\_matrix[0][int(index)-1] = Fraction(coeff[:-1] + "/1")

else:

self.coeff\_matrix[0][int(index)-1] = Fraction("-" + coeff[:-1] + "/1")

def objective\_maximize(self):

self.update\_objective\_function()

for row, column in enumerate(self.basic\_vars[1:]):

if self.coeff\_matrix[0][column] != 0:

self.coeff\_matrix[0] = add\_row(self.coeff\_matrix[0], multiply\_const\_row(-self.coeff\_matrix[0][column], self.coeff\_matrix[row+1]))

key\_column = min\_index(self.coeff\_matrix[0])

condition = self.coeff\_matrix[0][key\_column] < 0

print(Fore.WHITE + "Початкова симплекс-таблиця:" + Style.RESET\_ALL)

self.print\_simplex\_table()

print(Fore.WHITE + "\nДалі покроково показано розв'язання симплекс методом:" + Style.RESET\_ALL)

while condition is True:

key\_row = self.find\_key\_row(key\_column = key\_column)

self.basic\_vars[key\_row] = key\_column

pivot = self.coeff\_matrix[key\_row][key\_column]

self.normalize\_to\_pivot(key\_row, pivot)

self.make\_key\_column\_zero(key\_column, key\_row)

key\_column = min\_index(self.coeff\_matrix[0])

condition = self.coeff\_matrix[0][key\_column] < 0

self.print\_simplex\_table()

print("\n" + Fore.WHITE + 'Кінцева симплекс-таблиця має наступний вигляд:' + Style.RESET\_ALL)

self.print\_simplex\_table()

solution = {}

for i, var in enumerate(self.basic\_vars[1:]):

if var < self.num\_vars:

solution['y\_'+str(var+1)] = self.coeff\_matrix[i+1][-1]

for i in range(0, self.num\_vars):

if i not in self.basic\_vars[1:]:

solution['y\_'+str(i+1)] = Fraction("0/1")

count = 1

for i in range(self.num\_vars, len(self.coeff\_matrix[0]) - 1):

solution['x\_' + str(count)] = self.coeff\_matrix[0][i]

count += 1

return solution

def add\_row(row1, row2):

row\_sum = [0 for i in range(len(row1))]

for i in range(len(row1)):

row\_sum[i] = row1[i] + row2[i]

return row\_sum

def max\_index(row):

max\_i = 0

for i in range(0, len(row)-1):

if row[i] > row[max\_i]:

max\_i = i

return max\_i

def multiply\_const\_row(const, row):

mul\_row = []

for i in row:

mul\_row.append(const\*i)

return mul\_row

def min\_index(row):

min\_i = 0

for i in range(0, len(row)):

if row[min\_i] > row[i]:

min\_i = i

return min\_i

**Висновок:** в ході виконання лабораторної роботи було отримано теоретичні знання про матричці ігри, написано програму для вирішення матричної гри, яка шукає сідлову точку, спрощує матрицю шляхом знаходження домінуючих рядків та стовпці, та розв’язує отриману задачу лінійного програмування симплекс методом.