Міністерство освіти та науки України

Львівський національний університет імені Івана Франка

Факультет електроніки та

Комп’ютерних технологій

**Звіт**

Про виконання лабораторної роботи №1

“Прийняття рішень на прикладі розв'язку задачі про призначення”

Виконав:

Студент групи ФеІ-44

Сапанюк М.І.

Перевірив:

Мостова М.Р.

Львів 2022

**Мета:**

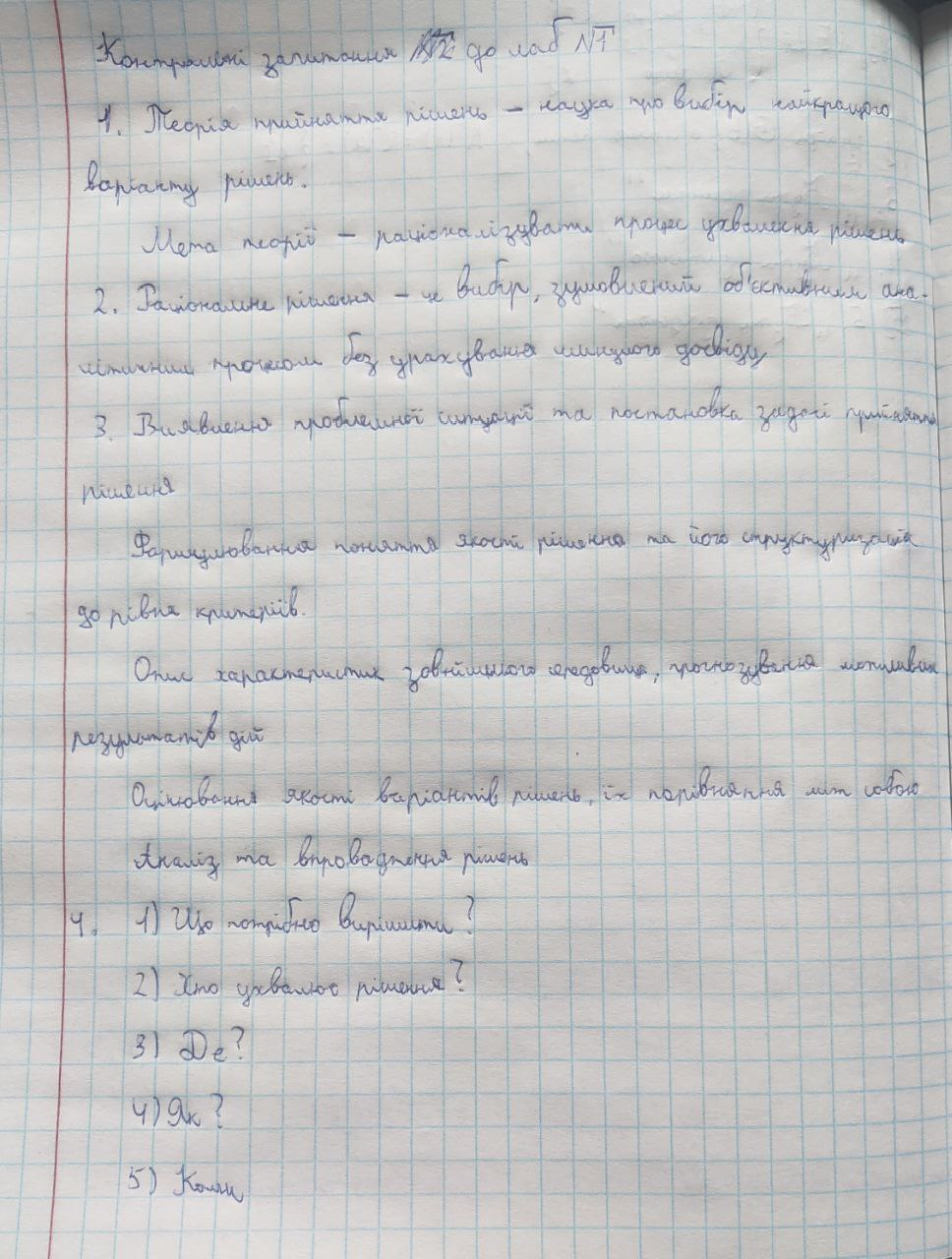
Постановка задачі про призначення. Засвоїти угорський алгоритм для розв’язку відкритої транспортної задачі.

**Хід роботи:**

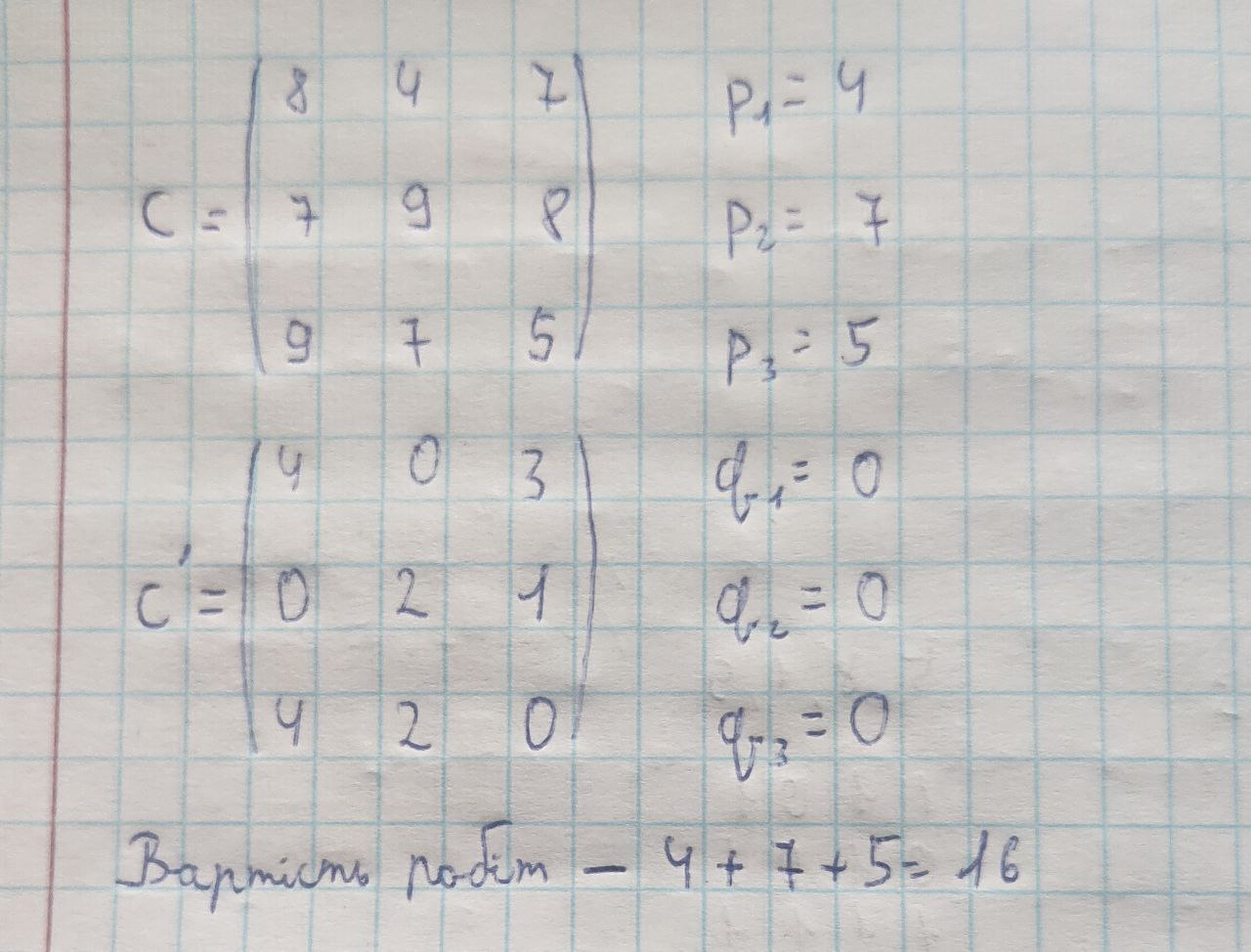
Завдання №1 Нехай маємо 3 різних робіт, кожну з яких може виконати будь-який з 3 працівників. Вартість виконання і-ої роботи j-м працівником дорівнює сij ( ці коефіцієнти формують матрицю витрат С ) грошових одиниць. Необхідно призначити кожного працівника на конкретну роботу так, щоб мінімізувати сумарну вартість робіт. Розв’язати задачу про призначення, яку задано матрицею С в зошиті. Написати програму, яка реалізує алгоритм розв’язку.

Примітка: Явний вигляд матриці брати зі ст. 166 (завдання 4.3) підручника Бартіш М. Я. рядки матриці – працівники, стовбці – види робіт згідно індивідуального варіанту

**Контрольні запитання:**

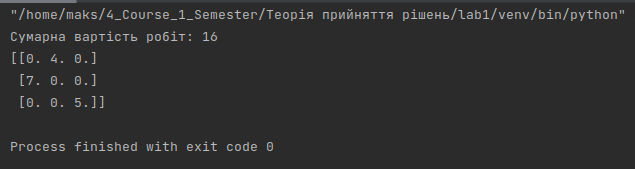


**Виконання завдання:**



import numpy as np  
  
  
def min\_zero\_row(zero\_mat, mark\_zero):  
 min\_row = [99999, -1]  
  
 for row\_num in range(zero\_mat.shape[0]):  
 if np.sum(zero\_mat[row\_num] == True) > 0 and min\_row[0] > np.sum(zero\_mat[row\_num] == True):  
 min\_row = [np.sum(zero\_mat[row\_num] == True), row\_num]  
  
 zero\_index = np.where(zero\_mat[min\_row[1]] == True)[0][0]  
 mark\_zero.append((min\_row[1], zero\_index))  
 zero\_mat[min\_row[1], :] = False  
 zero\_mat[:, zero\_index] = False  
  
  
def mark\_matrix(mat):  
 cur\_mat = mat  
 zero\_bool\_mat = (cur\_mat == 0)  
 zero\_bool\_mat\_copy = zero\_bool\_mat.copy()  
  
 marked\_zero = []  
 while (True in zero\_bool\_mat\_copy):  
 min\_zero\_row(zero\_bool\_mat\_copy, marked\_zero)  
  
 marked\_zero\_row = []  
 marked\_zero\_col = []  
 for i in range(len(marked\_zero)):  
 marked\_zero\_row.append(marked\_zero[i][0])  
 marked\_zero\_col.append(marked\_zero[i][1])  
  
 non\_marked\_row = list(set(range(cur\_mat.shape[0])) - set(marked\_zero\_row))  
  
 marked\_cols = []  
 check\_switch = True  
 while check\_switch:  
 check\_switch = False  
 for i in range(len(non\_marked\_row)):  
 row\_array = zero\_bool\_mat[non\_marked\_row[i], :]  
 for j in range(row\_array.shape[0]):  
 if row\_array[j] == True and j not in marked\_cols:  
 marked\_cols.append(j)  
 check\_switch = True  
  
 for row\_num, col\_num in marked\_zero:  
 if row\_num not in non\_marked\_row and col\_num in marked\_cols:  
 non\_marked\_row.append(row\_num)  
 check\_switch = True  
 marked\_rows = list(set(range(mat.shape[0])) - set(non\_marked\_row))  
  
 return (marked\_zero, marked\_rows, marked\_cols)  
  
  
def adjust\_matrix(mat, cover\_rows, cover\_cols):  
 cur\_mat = mat  
 non\_zero\_element = []  
  
 for row in range(len(cur\_mat)):  
 if row not in cover\_rows:  
 for i in range(len(cur\_mat[row])):  
 if i not in cover\_cols:  
 non\_zero\_element.append(cur\_mat[row][i])  
 min\_num = min(non\_zero\_element)  
  
 for row in range(len(cur\_mat)):  
 if row not in cover\_rows:  
 for i in range(len(cur\_mat[row])):  
 if i not in cover\_cols:  
 cur\_mat[row, i] = cur\_mat[row, i] - min\_num  
 for row in range(len(cover\_rows)):  
 for col in range(len(cover\_cols)):  
 cur\_mat[cover\_rows[row], cover\_cols[col]] = cur\_mat[cover\_rows[row], cover\_cols[col]] + min\_num  
 return cur\_mat  
  
  
def hungarian\_algorithm(mat):  
 dim = mat.shape[0]  
 cur\_mat = mat  
  
 for row\_num in range(mat.shape[0]):  
 cur\_mat[row\_num] = cur\_mat[row\_num] - np.min(cur\_mat[row\_num])  
  
 for col\_num in range(mat.shape[1]):  
 cur\_mat[:, col\_num] = cur\_mat[:, col\_num] - np.min(cur\_mat[:, col\_num])  
 zero\_count = 0  
 while zero\_count < dim:  
 ans\_pos, marked\_rows, marked\_cols = mark\_matrix(cur\_mat)  
 zero\_count = len(marked\_rows) + len(marked\_cols)  
  
 if zero\_count < dim:  
 cur\_mat = adjust\_matrix(cur\_mat, marked\_rows, marked\_cols)  
  
 return ans\_pos  
  
  
def ans\_calculation(mat, pos):  
 total = 0  
 ans\_mat = np.zeros((mat.shape[0], mat.shape[1]))  
 for i in range(len(pos)):  
 total += mat[pos[i][0], pos[i][1]]  
 ans\_mat[pos[i][0], pos[i][1]] = mat[pos[i][0], pos[i][1]]  
 return total, ans\_mat  
  
  
def main():  
 cost\_matrix = np.array([[8, 4, 7],  
 [7, 9, 8],  
 [9, 7, 5],])  
 ans\_pos = hungarian\_algorithm(cost\_matrix.copy())  
 ans, ans\_mat = ans\_calculation(cost\_matrix, ans\_pos)  
  
 print(f"Сумарна вартість робіт: {ans:.0f}\n{ans\_mat}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Результат виконання:**



**Висновок:**

На цій лабораторній роботі я дослідив та запрограмував угорський алгоритм для розв’язку відкритої транспортної задачі.