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:

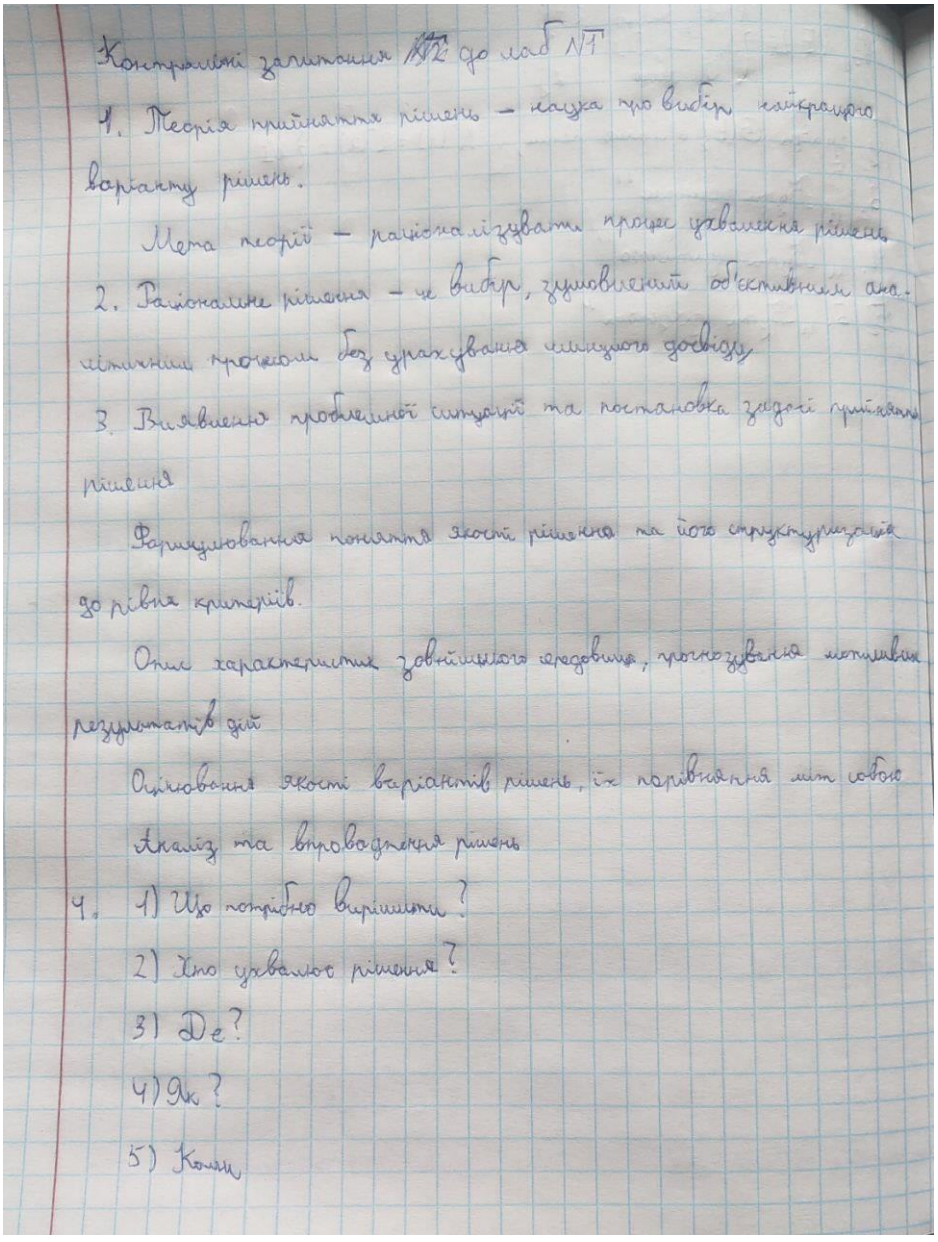
Постановка задачі про призначення. Засвоїти угорський алгоритм для розв’язку відкритої транспортної задачі.

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:


Контрольні запитання №2 до лаб №1

1. Теорія прийняття рішень – наука про вибір найкращого варіанту рішень.

Мета теорії – раціоналізувати процес ухвалення рішень

2. Раціональне рішення – це вибір, здійснений об'єктивним аналітичним процесом без урахування минулого досвіду

3. Виявлення проблемної ситуації та постановка задачі прийняття рішення

Формулювання поняття якості рішення та його структуризація до рівня критеріїв.

Опис характеристик зовнішнього середовища, прогнозування можливих результатів дій

Оцінювання якості варіантів рішень, їх порівняння між собою

Аналіз та впровадження рішень

4. 1) Що потрібно вирішити?

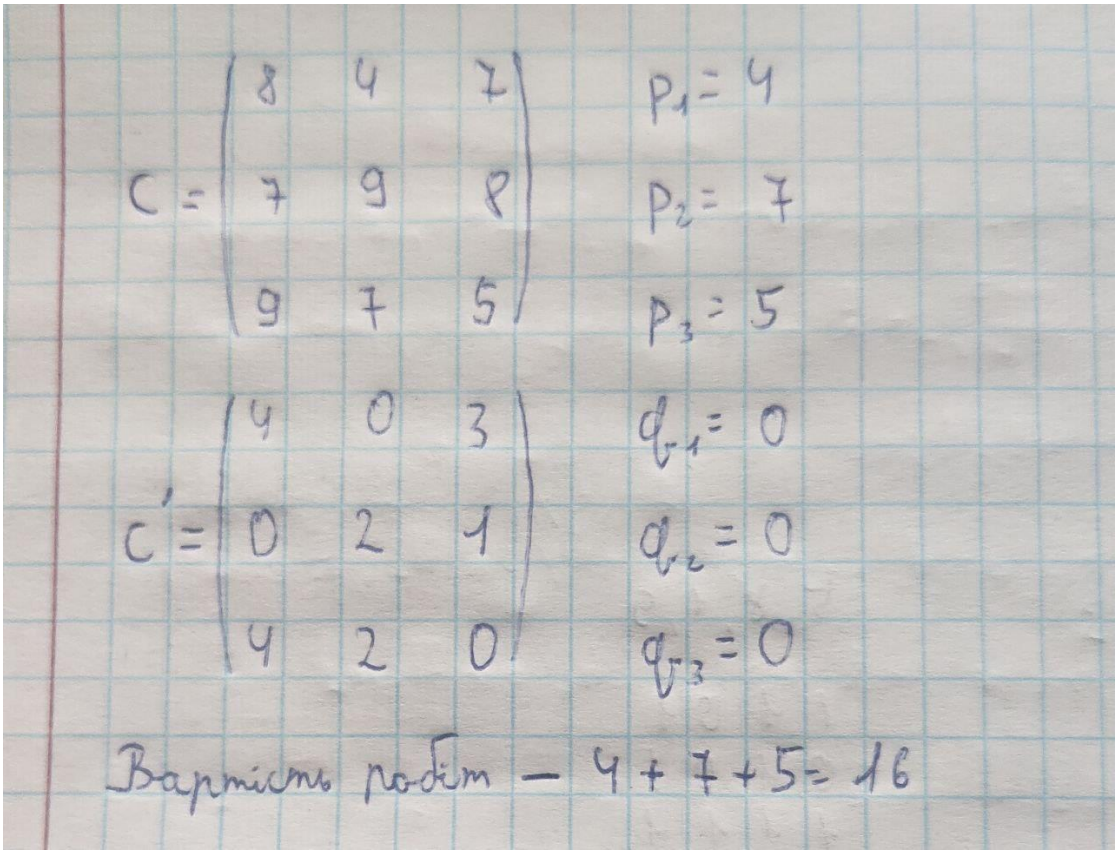
2) Хто ухвалює рішення?

3) Де?

4) Як?

5) Коли

Виконання завдання:




```python
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:

```
"/home/maks/4_Course_1_Semester/Теорія прийняття рішень/lab1/venv/bin/python"
Сумарна вартість робіт: 16
[[0. 4. 0.]
 [7. 0. 0.]
 [0. 0. 5.]]

Process finished with exit code 0
```

Висновок:

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