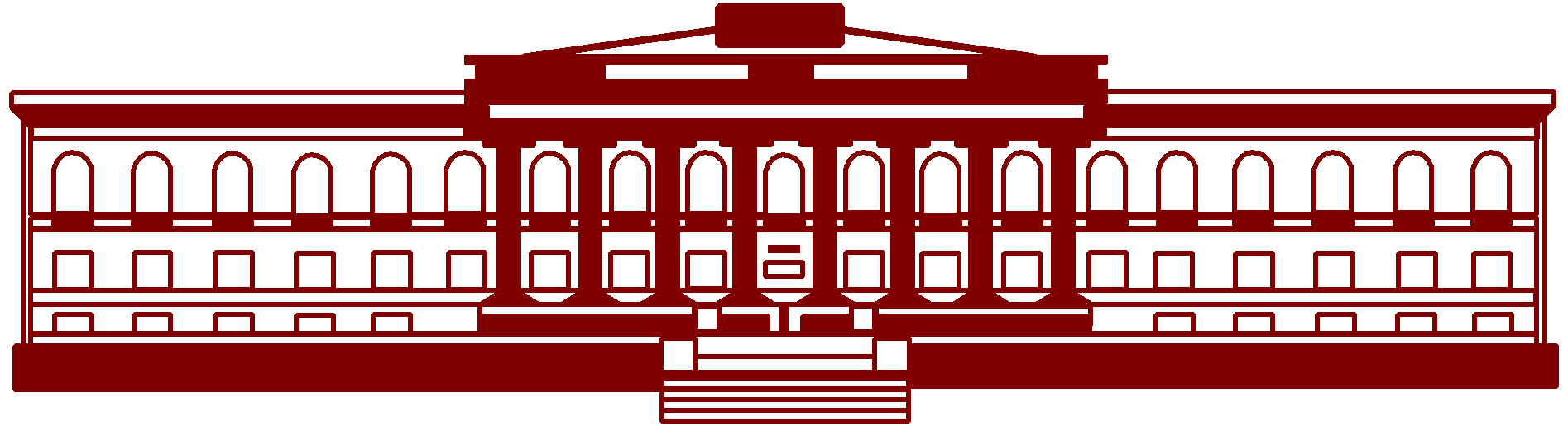
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

****

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра прикладних інформаційних систем**

**Звіт до лабораторної роботи №7**

**з курсу**

**«Системний аналіз та теорія прийняття рішень»**

*Студентa 3 курсу*

*групи ПП-32*

*спеціальності 122 «Комп'ютерні науки»*

*ОП «Прикладне програмування»*

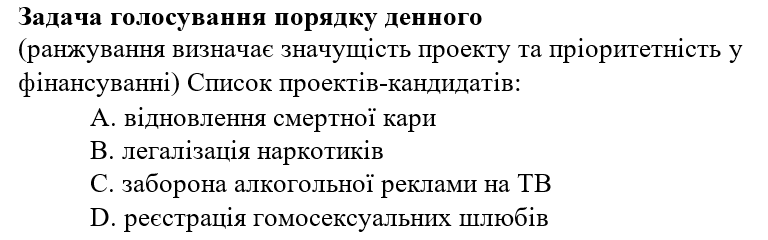
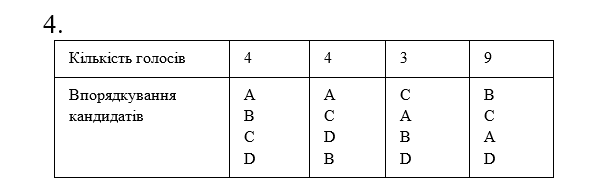
*Кондратова Івана Андрійовича*

*Викладач:*

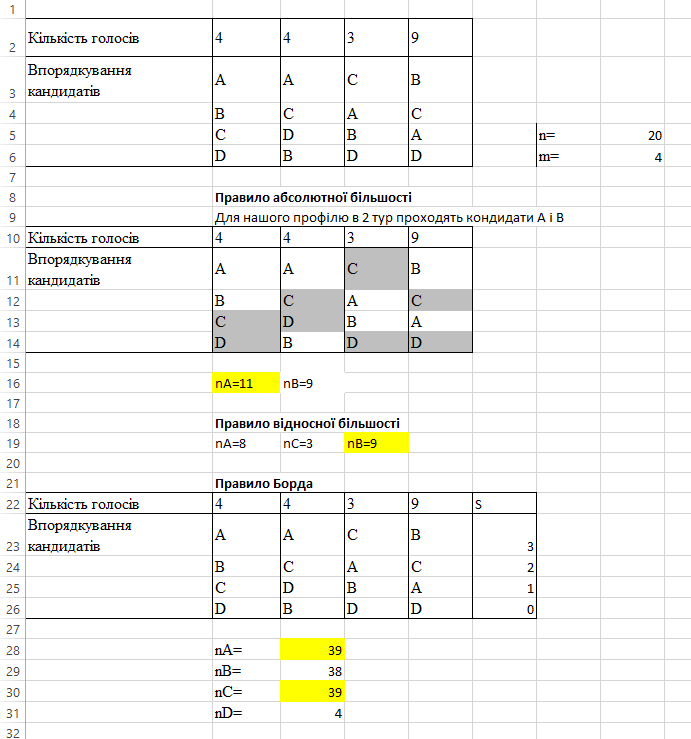
Плескач В.Л.

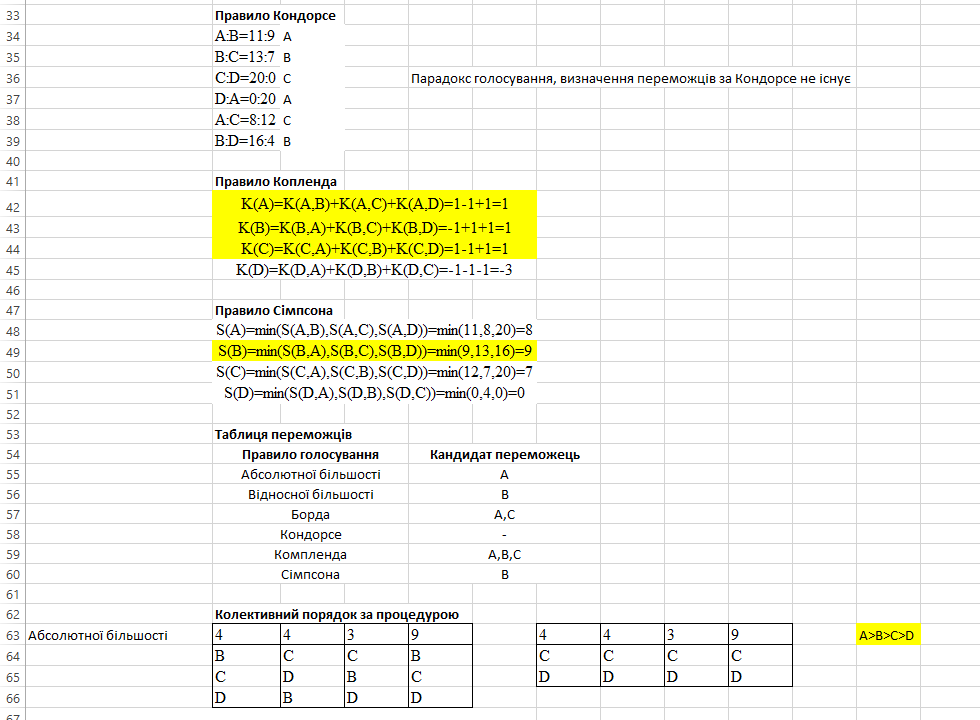
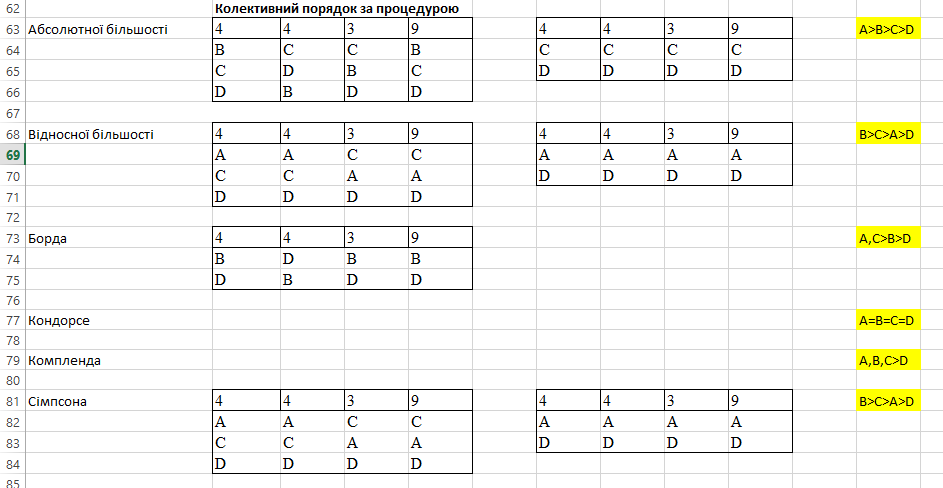
**Київ – 2023**

**Тема:** методи голосування

**Мета:** вивчення методів голосування і дослідження їх властивостей; опанування методикою обробки профілів колективного голосування, знаходження переможця та відновлення колективного ранжування  

**Розв’язання в Excel**



**Розв’язання в Python**

import numpy as np

from sklearn.preprocessing import LabelBinarizer

def relative\_majority(e, votes):

    return (e[0] \* votes).sum(axis=1)

def encoded(matrix, label\_binarizer):

    encoded = label\_binarizer.fit\_transform(matrix.flatten())

    encoded = encoded.reshape(matrix.shape + (4,))

    encoded = encoded.transpose((0, 2, 1))

    return encoded

def relative(labels, e, votes):

    relative\_majority\_score = relative\_majority(e, votes)

    relative\_majority\_result = labels[relative\_majority\_score.argmax()]

    return relative\_majority\_result

def absolute(labels, matrix, votes, encoded, label\_binarizer):

    relative\_majority\_score = (encoded[0] \* votes).sum(axis=1)

    sorted\_result = np.sort(relative\_majority\_score)[-2:]

    indices = np.where(np.isin(relative\_majority\_score, sorted\_result))[0]

    second\_indicies = np.argmax(np.isin(matrix, labels[indices]), axis=0)

    top2\_vector = label\_binarizer.transform(

        matrix[second\_indicies, np.arange(matrix.shape[1])]).T \* votes

    absolute\_score = top2\_vector.sum(axis=1)

    absolute\_result = labels[absolute\_score.argmax()]

    return absolute\_result

def borda(encoded, votes, labels):

    mul\_array = encoded \* votes

    arr\_score = mul\_array \* np.array([3, 2, 1, 0])[:, np.newaxis, np.newaxis]

    bord\_score = arr\_score.sum(axis=2).sum(axis=0)

    bord\_result = labels[bord\_score.argmax()]

    return bord\_result

def condorse(matrix, votes, labels):

    unique\_chars = np.unique(matrix)

    sorted\_chars = np.sort(unique\_chars)

    char\_to\_num = {char: i for i, char in enumerate(sorted\_chars)}

    num\_matrix = np.vectorize(char\_to\_num.get)(matrix)

    matrix\_cond = np.zeros((4, 4))

    for i, col in enumerate(num\_matrix.T):

        for x in range(col.shape[0]):

            for y in range(x+1, col.shape[0]):

                matrix\_cond[col[x]][col[y]] += votes[i]

    for i, e1 in enumerate(labels):

        winner = True

        for j in range(labels.shape[0]):

            if i != j:

                if matrix\_cond[i][j] < matrix\_cond[j][i]:

                    winner = False

        if winner:

            matrix\_result = e1

            return matrix\_result

def coplend(matrix, votes, labels):

    unique\_chars = np.unique(matrix)

    sorted\_chars = np.sort(unique\_chars)

    char\_to\_num = {char: i for i, char in enumerate(sorted\_chars)}

    num\_matrix = np.vectorize(char\_to\_num.get)(matrix)

    matrix\_cond = np.zeros((4, 4))

    for i, col in enumerate(num\_matrix.T):

        for x in range(col.shape[0]):

            for y in range(x+1, col.shape[0]):

                matrix\_cond[col[x]][col[y]] += votes[i]

    cop\_score = np.zeros(matrix.shape[0])

    for i, e1 in enumerate(labels):

        for j in range(labels.shape[0]):

            if i != j:

                if matrix\_cond[i][j] < matrix\_cond[j][i]:

                    cop\_score[i] -= 1

                elif matrix\_cond[i][j] > matrix\_cond[j][i]:

                    cop\_score[i] += 1

    cop\_result = labels[cop\_score.argmax()]

    return cop\_result

def simpson(matrix, votes, labels):

    unique\_chars = np.unique(matrix)

    sorted\_chars = np.sort(unique\_chars)

    char\_to\_num = {char: i for i, char in enumerate(sorted\_chars)}

    num\_matrix = np.vectorize(char\_to\_num.get)(matrix)

    matrix\_cond = np.zeros((4, 4))

    for i, col in enumerate(num\_matrix.T):

        for x in range(col.shape[0]):

            for y in range(x+1, col.shape[0]):

                matrix\_cond[col[x]][col[y]] += votes[i]

    data\_no\_zeros = np.where(matrix\_cond == 0, np.nan, matrix\_cond)

    simpson\_score = np.nanmin(data\_no\_zeros, axis=1)

    simpson\_result = labels[simpson\_score.argmax()]

    return simpson\_result

def main():

    label\_binarizer = LabelBinarizer()

    matrix = np.array([['A', 'A', 'C', 'B'],

                       ['B', 'C', 'A', 'C'],

                       ['C', 'D', 'B', 'A'],

                       ['D', 'B', 'D', 'D'],])

    votes = np.array([4, 4, 3, 9])

    e = encoded(matrix, label\_binarizer)

    labels = np.array(['A', 'B', 'C', 'D'])

    a = absolute(labels, matrix, votes, e, label\_binarizer)

    print(a)

    r = relative(labels, e, votes)

    print(r)

    borda\_res = borda(e, votes, labels)

    print(borda\_res)

    condorse\_res = condorse(matrix, votes, labels)

    print(condorse\_res)

    kop\_res = coplend(matrix, votes, labels)

    print(kop\_res)

    simpson\_res = simpson(matrix, votes, labels)

    print(simpson\_res)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Контрольні питання**

**1. Які аксіоми визначають апріорі «розумні» «логічні» вимоги до правил голосування?**

Аксіома Порядку: Всі виборці повинні мати можливість виразити свої вподобання чітко і однозначно.

Аксіома Незалежності: Вплив на результат голосування не повинен залежати від порядку висловлювання виборців.

Аксіома Беззмінності Альтернатив: Додавання або вилучення альтернатив, які не пов'язані з перемогою або поразкою певних альтернатив, не повинно впливати на вибір між іншими альтернативами.

**2. Які аксіоми визначають властивості голосування?**

Аксіома Відсутності Диктатора: Відсутність такої особи (диктатора), яка однозначно визначає результат голосування.

Аксіома Універсальності: Кожен виборець має право голосувати за будь-яку альтернативу.

**3. Яке правило голосування відповідає критерію утилітаризму, а яке критерію егалітаризму функції колективної корисності? В чому полягає суть цих критеріїв?**

 Утилітаризм: Відповідне правило - голосування більшістю. Це правило максимізує загальну користь або утилітарний сумарний благодійний внесок.

Егалітаризм функції колективної корисності: Відповідне правило - голосування за принципом "кожен за одного, один за всіх". Всі голосують за одну альтернативу, яка максимізує користь найменш благополучного члена суспільства.

**4. Які є парадокси голосування? Які можуть бути маніпуляції при колективному голосуванні?**

Парадокси голосування та маніпуляції:

Парадокс вибору: Існування ситуації, коли колективне рішення суперечить індивідуальним виборам.

Маніпуляції: Це включає стратегічне голосування або маніпулювання іншими аспектами процесу голосування для досягнення бажаного результату.

**5. Які правила голосування застосовуються в ситуаціях: спортивних змагань, вибору депутатів, вибору президента?**

Спортивні змагання: Зазвичай використовується система кругового турніру або система вилучення (елімінації).

Вибори депутатів: Часто застосовується система мажоритарного голосування або пропорційна система представництва.

Вибори президента: Зазвичай використовується система голосування "один за всіх" (перша альтернатива, яка отримує більше половини голосів, перемагає).

**Висновок:** виконуючи цю лабораторну роботу, явивчив методи голосування і дослідження їх властивостей; опанував методику обробки профілів колективного голосування, знаходження переможця та відновлення колективного ранжування