МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет імені Михайла Драгоманова

Факультет математики, інформатики та фізики

Кафедра інформаційних технологій та програмування

**Звіт**

з лабораторної роботи №9

«Класичні алгоритми опрацювання масивів»

з дисципліни «Програмування»

Виконав:

студент ІІІ курсу групи 31І

Дзюба В.І.

Перевірила:

викладач Устименко О.Б.

Оцінка\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_

**Київ – 2024**

**Зміст**

[Мета роботи 3](#_Toc168149373)

[Постановка задачі 4](#_Toc168149374)

[Основна частина 5](#_Toc168149375)

[2.1 Опис вхідних та вихідних даних 5](#_Toc168149376)

[2.2 Блок-схема 6](#_Toc168149377)

[Висновки 7](#_Toc168149378)

[Додатки 8](#_Toc168149379)

# Мета роботи

Метою цієї лабораторної роботи є набуття навичок реалізації та застосування класичних алгоритмів опрацювання масивів на прикладі алгоритму бінарного пошуку, зокрема, ознайомлення з принципами роботи бінарного пошуку, генерування двовимірних масивів з випадковими елементами, реалізація алгоритму бінарного пошуку для знаходження заданого елементу в стовпцях двовимірного масиву, перевірка правильності роботи програми на прикладах з різними розмірами масивів і значеннями елементів, закріплення знань з основ алгоритмів і структур даних, покращення навичок програмування, а також ефективне використання алгоритмів для опрацювання масивів.

# Постановка задачі

Варіант №12

**Завдання**

Напишіть програму, яка реалізує класичний алгоритм бінарного пошуку заданого елементу у стовпцях двовимірного масиву. Розмірність масиву та всі елементи генеруються за допомогою випадкових чисел.

# Основна частина

## 2.1 Опис вхідних та вихідних даних

**Вхідні дані:**

1. *rows, cols*: Визначаються випадковим чином, встановлюють розміри двовимірного масиву (матриці). Випадкове ціле число від 1 до 10 для кількості рядків і стовпців.

2. *matrix*: Двовимірний масив розміром (*rows* x *cols*), заповнений випадковими цілими числами від 1 до 100.

3. *target*: Ціле число, введене користувачем, яке необхідно знайти у стовпцях матриці.

**Вихідні дані:**

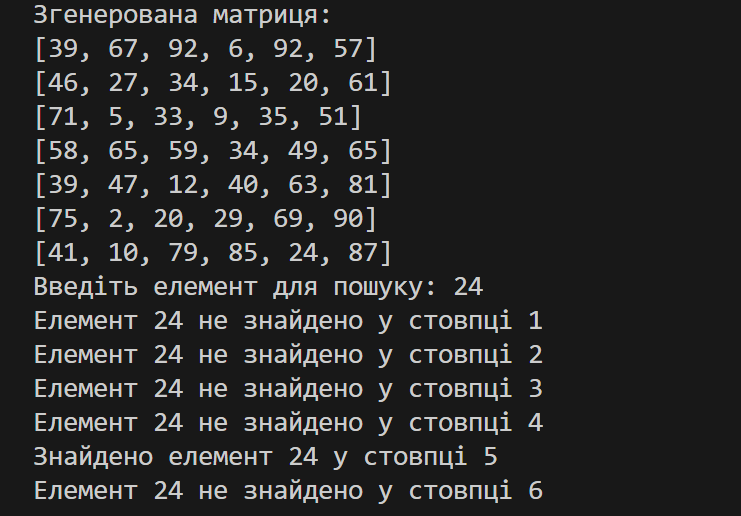
1. *matrix*: Виведена згенерована матриця, розміром (*rows* x *cols*),

2. *found*: Результат пошуку заданого елемента у кожному стовпці матриці:

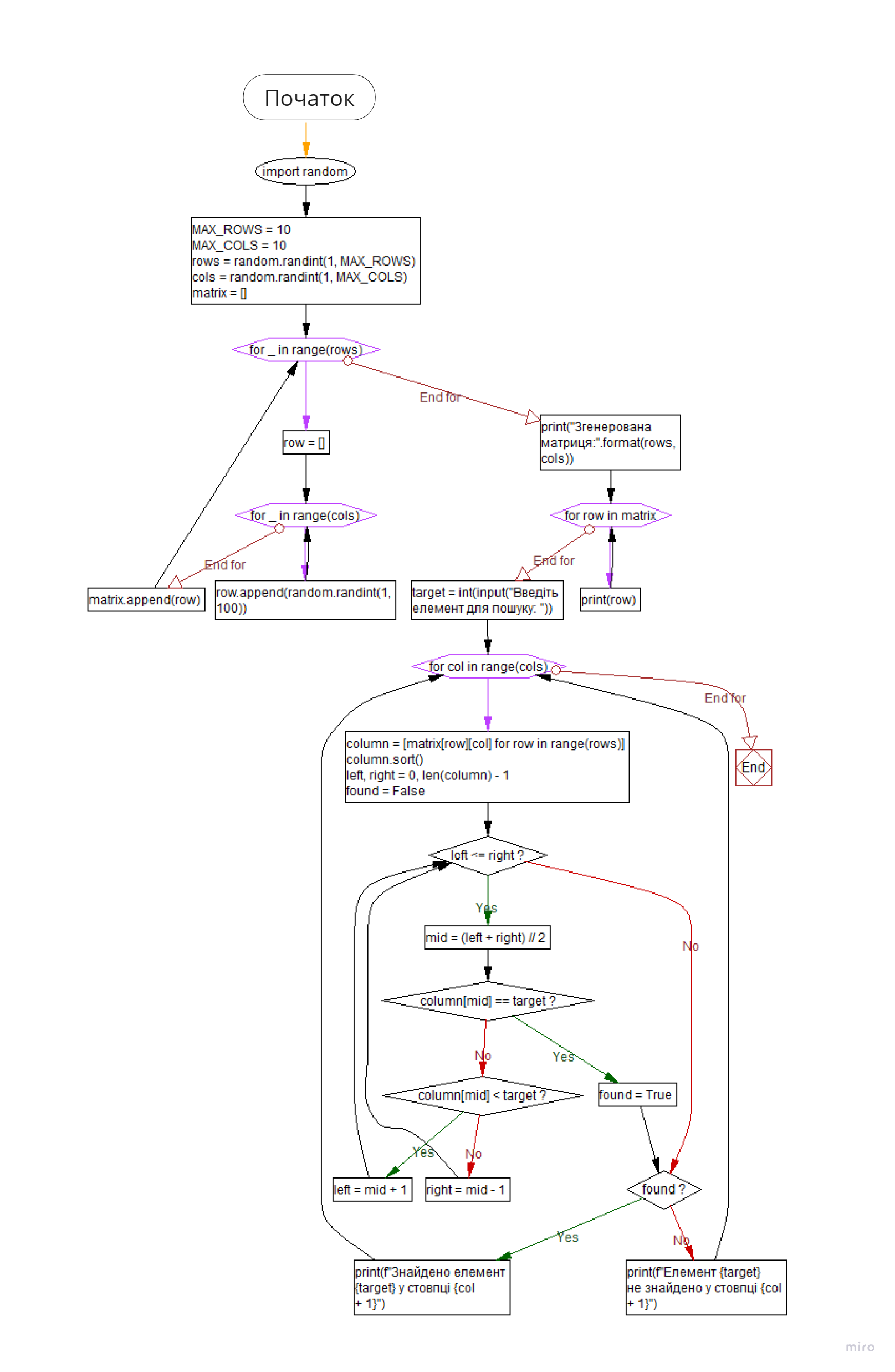
- Повідомлення "Знайдено елемент {target} у стовпці {col + 1}", якщо елемент знайдено у відповідному стовпці.

- Повідомлення "Елемент {target} не знайдено у стовпці {col + 1}", якщо елемент не знайдено у відповідному стовпці.

**Приклад виконаної програми**:



## 2.2 Блок-схема



# Висновки

Виконана лабораторна робота дозволила закріпити знання з алгоритмів обробки масивів, зокрема бінарного пошуку, шляхом генерування випадкових двовимірних масивів, реалізації алгоритму для пошуку заданого елементу в стовпцях масиву, та покращення навичок програмування і роботи з даними на Python.

# Додатки

# Варіант 12

#

# Напишіть програму, яка реалізує класичний алгоритм бінарного

# пошуку заданого елементу у стовпцях двовимірного масиву.

# Розмірність масиву та всі елементи генеруються за допомогою випадкових чисел.

#

# Дзюба Владислав

import random

# Максимальні розміри матриці

MAX\_ROWS = 10

MAX\_COLS = 10

# Визначаємо розміри матриці випадковим чином

rows = random.randint(1, MAX\_ROWS)

cols = random.randint(1, MAX\_COLS)

# Генеруємо двовимірний масив із випадкових чисел

matrix = []

for \_ in range(rows):

    row = []

    for \_ in range(cols):

        row.append(random.randint(1, 100))

    matrix.append(row)

# Виводимо згенеровану матрицю

print("Згенерована матриця:".format(rows, cols))

for row in matrix:

    print(row)

# Заданий елемент для пошуку

target = int(input("Введіть елемент для пошуку: "))

# Шукаємо заданий елемент у стовпцях матриці

for col in range(cols):

    # Отримуємо стовпець як список

    column = [matrix[row][col] for row in range(rows)]

    # Сортуємо стовпець перед бінарним пошуком

    column.sort()

    # Бінарний пошук у стовпці

    left, right = 0, len(column) - 1

    found = False

    while left <= right:

        mid = (left + right) // 2

        if column[mid] == target:

            found = True

            break

        elif column[mid] < target:

            left = mid + 1

        else:

            right = mid - 1

    if found:

        print(f"Знайдено елемент {target} у стовпці {col + 1}")

    else:

        print(f"Елемент {target} не знайдено у стовпці {col + 1}")

Результат:

