Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

**Лабораторна робота №2**

з дисципліни:

«Мультипарадигменне програмування»

Виконав:

студент групи ІС-23

Шимків Мирослав

Київ 2025

**ЛАБОРАТОРНАЯ РОБОТА №2**

**Завдання:** на мові функціонального програмування реалізувати перетворення чисельного ряду до лінгвістичного ланцюжка за певним розподілом ймовірностей потрапляння значень до інтервалів з подальшою побудовою матриці передування.

**Вхідні данні**: чисельний ряд, вид розподілу ймовірностей, потужність алфавіту.

**Вихідні дані**: лінгвістичний ряд та матриця передування.

**Мова програмування**: Racket, Common Lisp, **Clojure**

**Варіант 24**

(завдання 4 відповідно)

Мій варіант 4, отже, я обрав Хі-розподіл і мову програмування Clojure

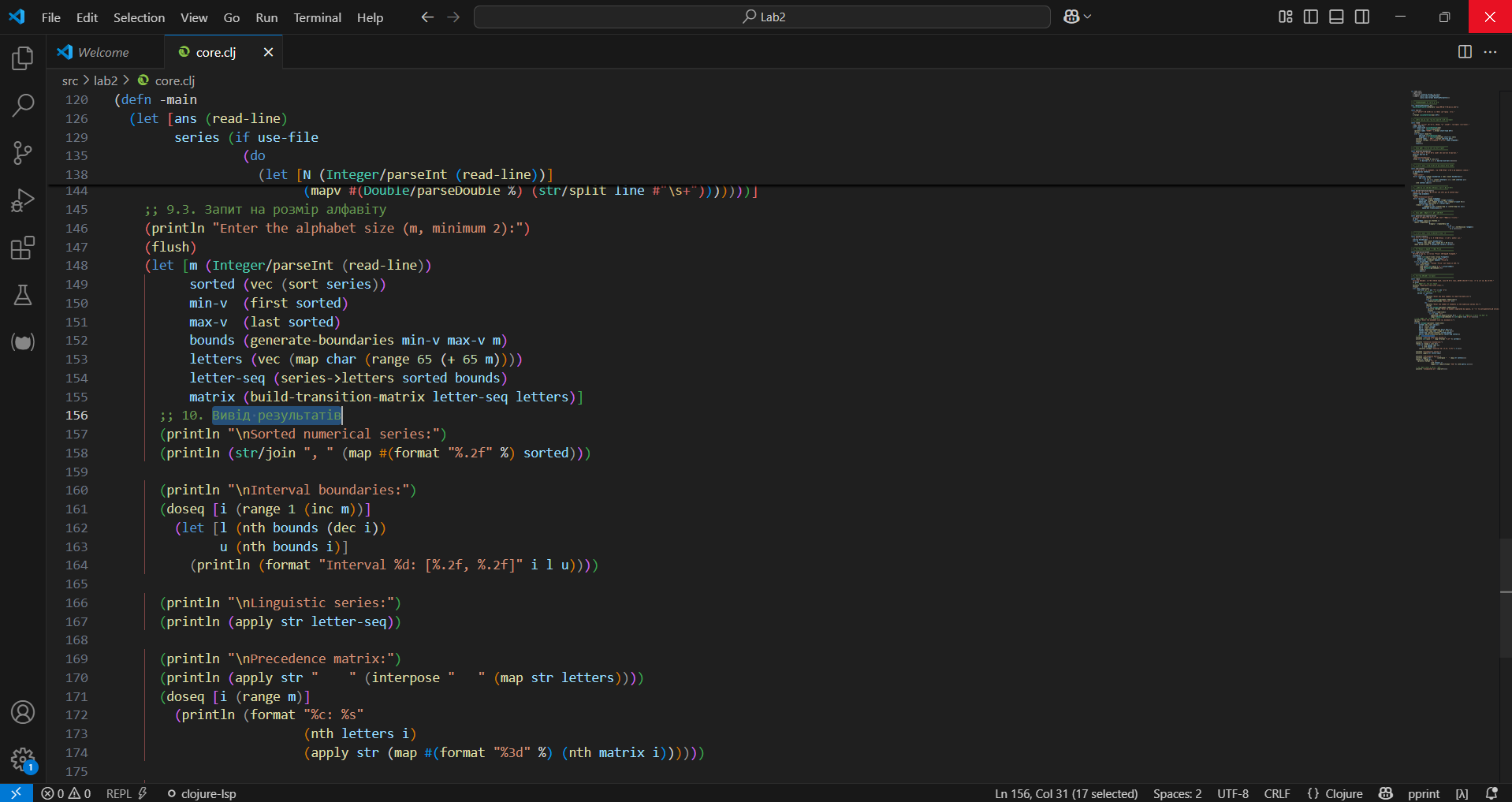
**Виконання роботи**

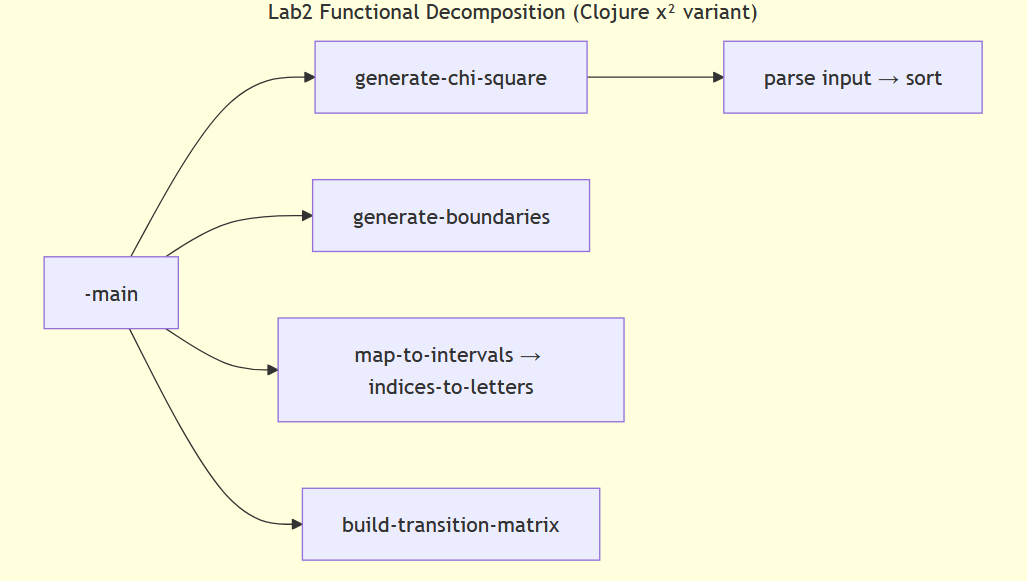
**Нижче наведено логічні блоки програми:**

1. Ініціалізація форматтера часу
2. Універсальна обгортка для заміру часу виконання (timed)
3. Генерація рівномірних меж інтервалів (generate-boundaries)
4. Перетворення числа у літеру за межами інтервалів (num->letter)
5. Побудова матриці передування, частот переходів (build-transition-matrix)
6. Генерація випадкового χ²-розподілу (generate-chi-square-series)
7. Перетворення чисел у лінгвістичний ряд (series->letters)
8. Зчитування з файлу стовпця Price (read-price-column)
9. Головна функція виконання
   1. Запит про джерело даних
   2. Зчитування серії чисел
   3. Запит на розмір алфавіту
10. Вивід результатів
11. Завершення з міткою часу

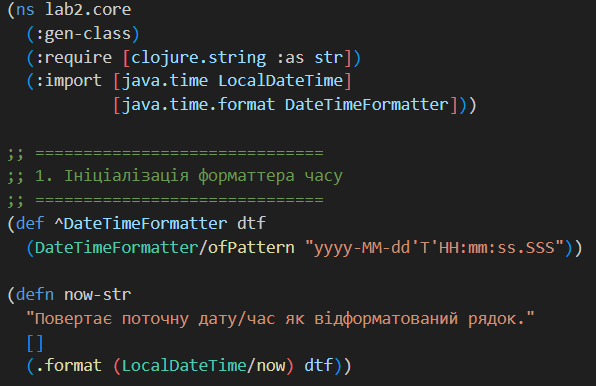
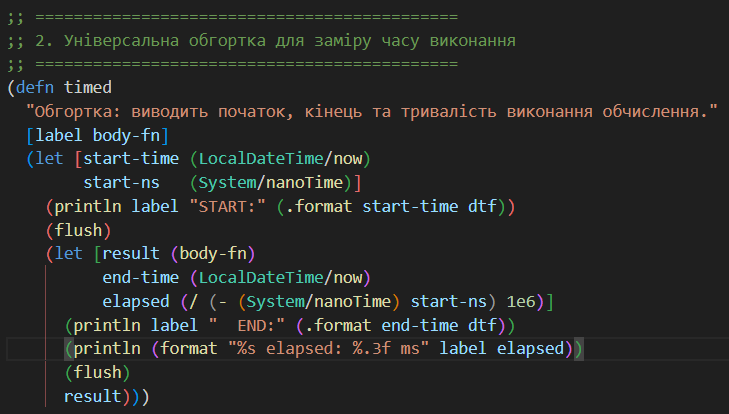
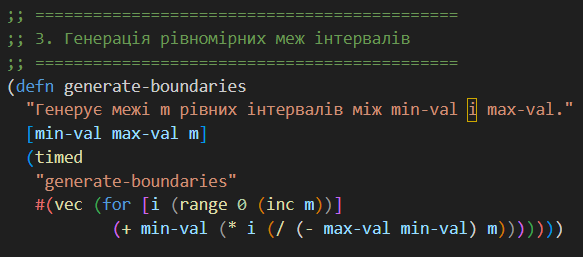
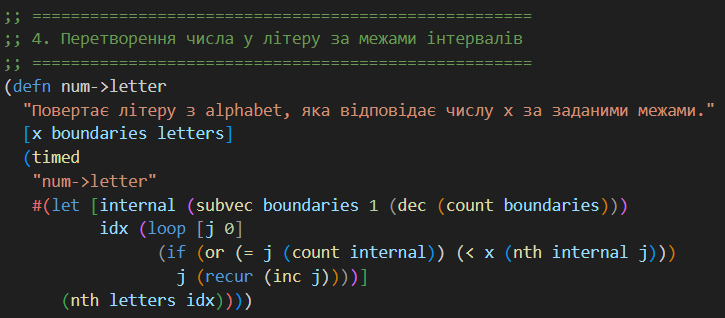
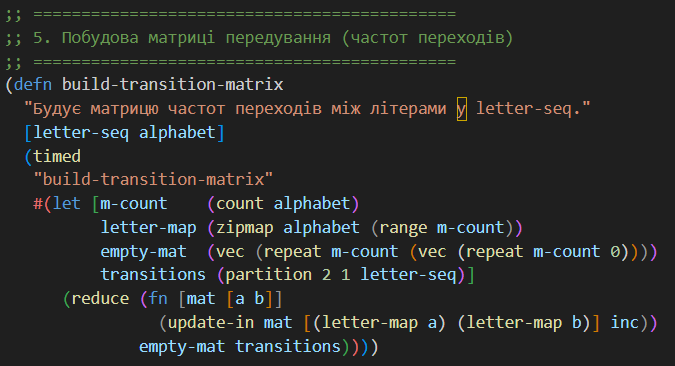
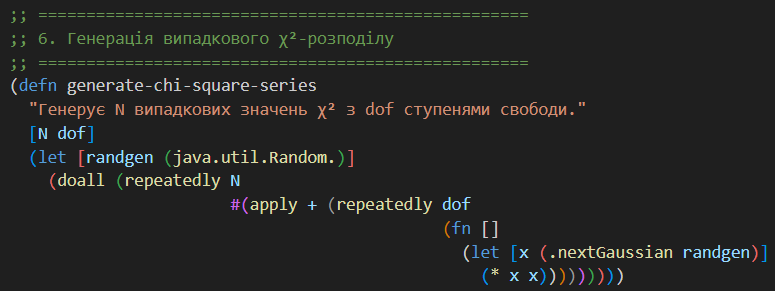
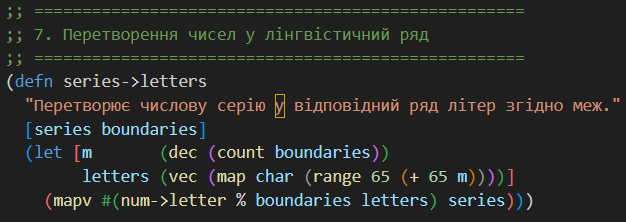
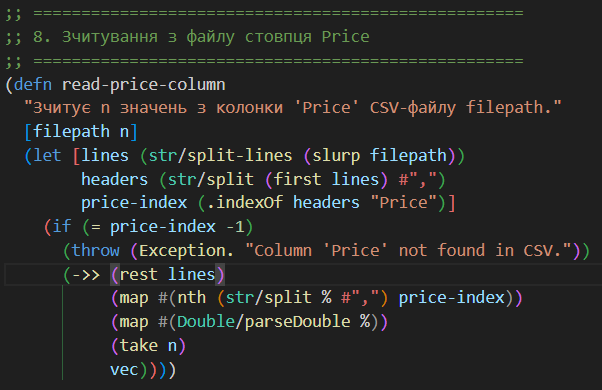
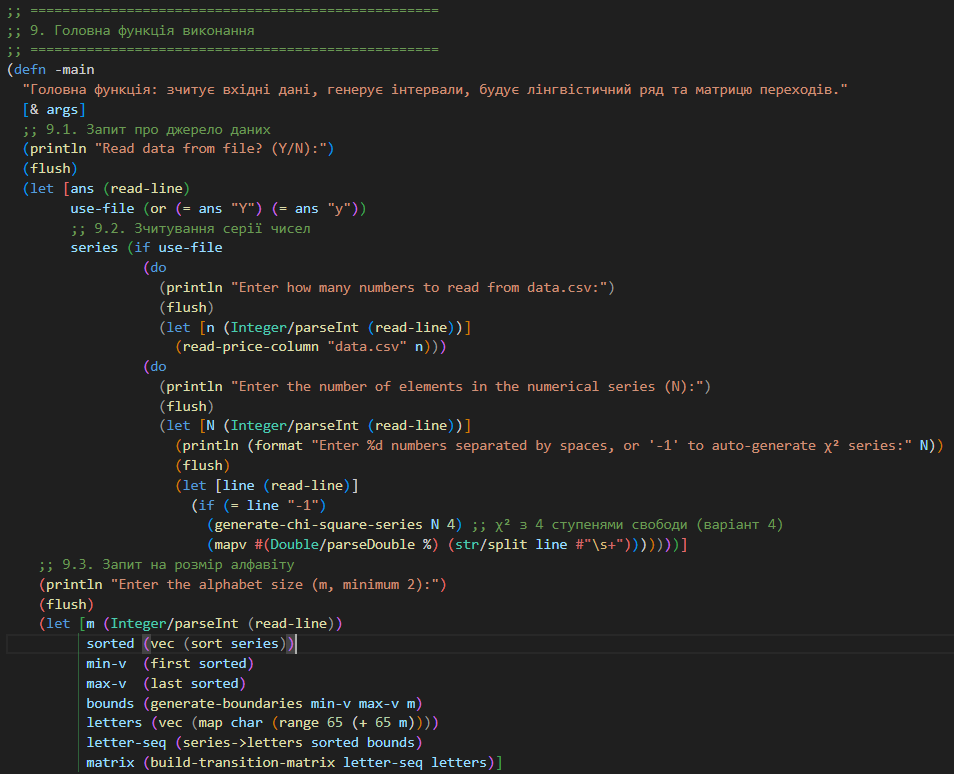
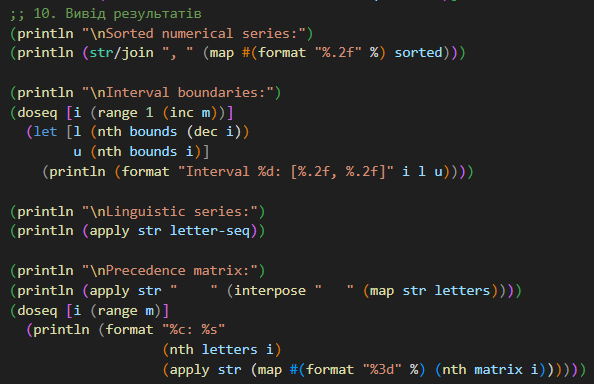
**Давайте розглянемо код для кожного пункту, в якому наведено коментарі українською мовою для кращого розуміння:**

Інтерфейс Visual Studio Code for Clojure для запуску програми



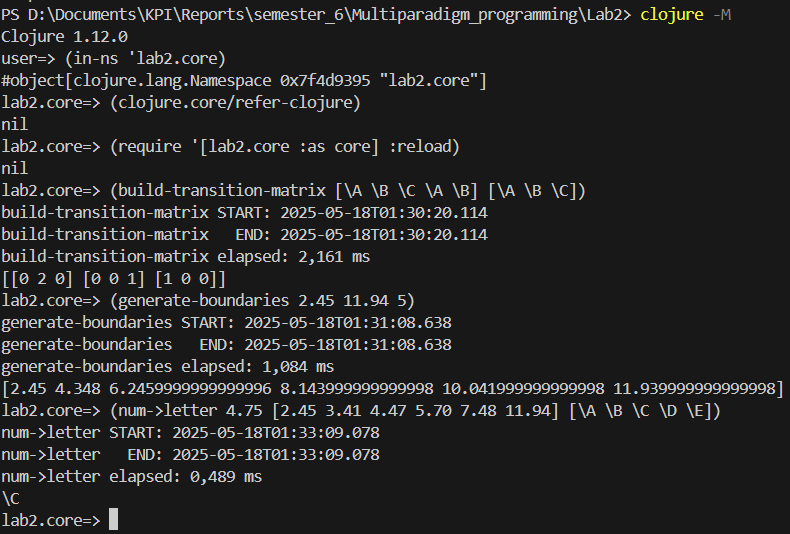
Функціональна декомпозиція і діаграма виконання програми:  


Етапи:

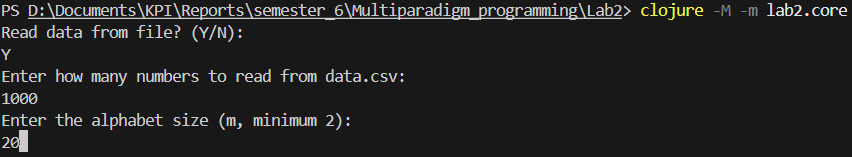
1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 

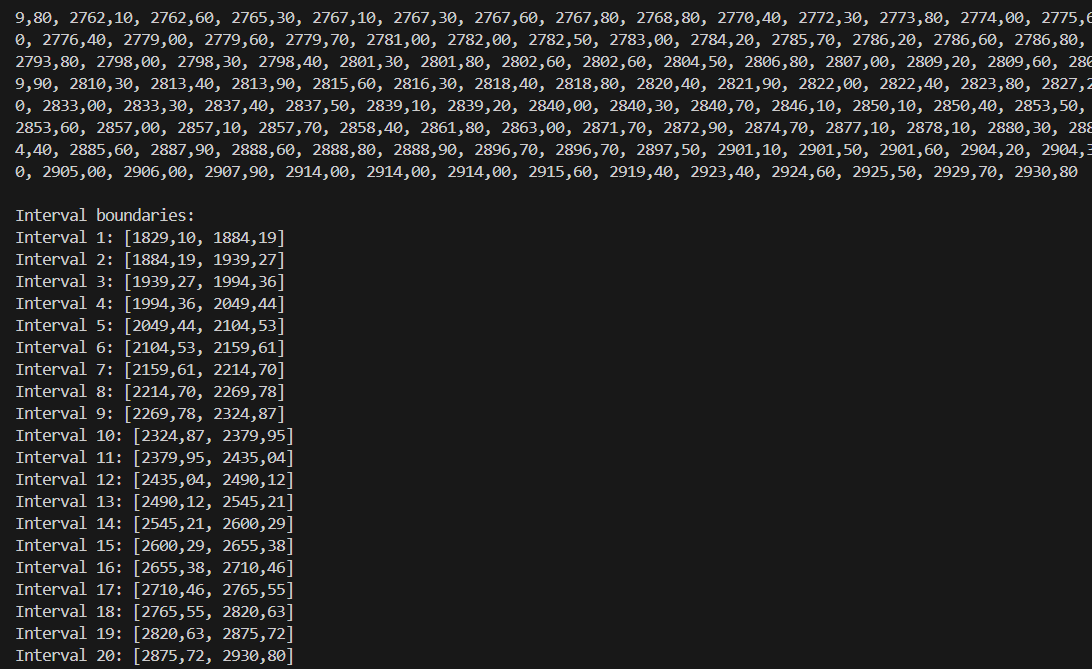
**Вхідні данні і результат виконання програми:**

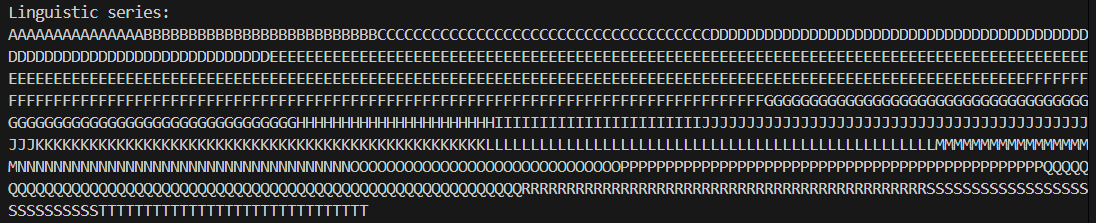
Скриншоти викликів у REPL, разом з датами і часом виконання:

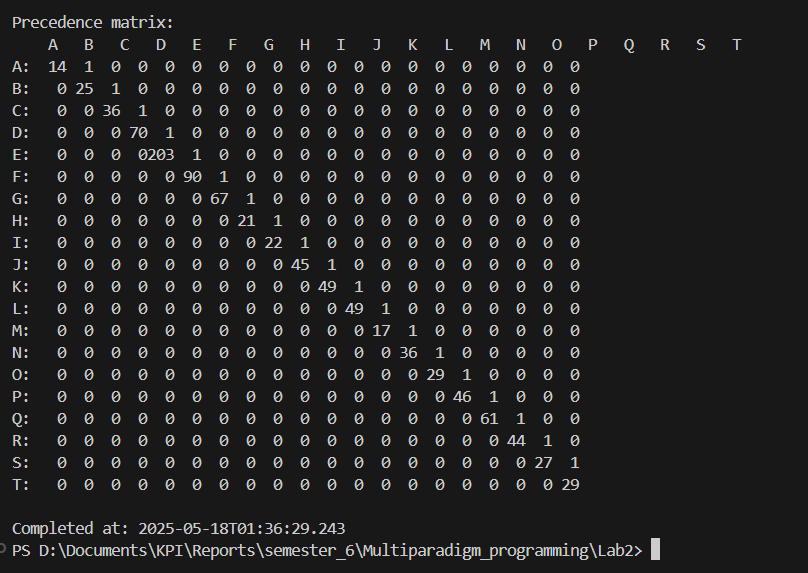


Читання з файлу часового ряду “B-C-D-E-S&P 500 Historical Data” з архіву, а саме поля Price:

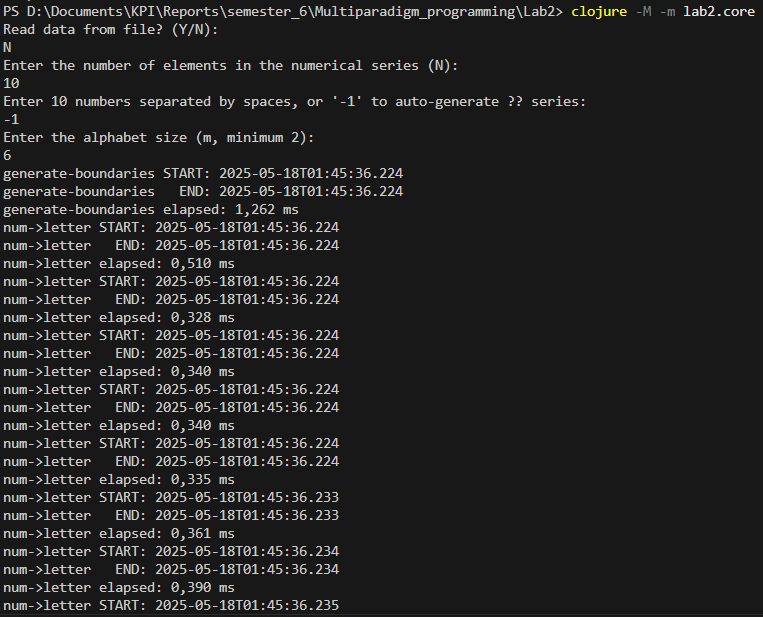


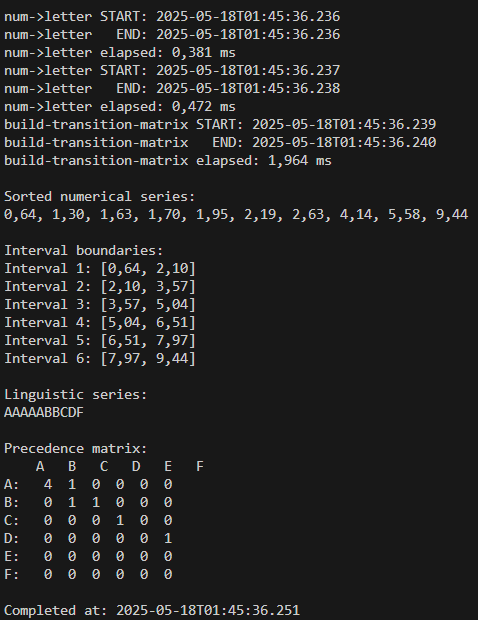




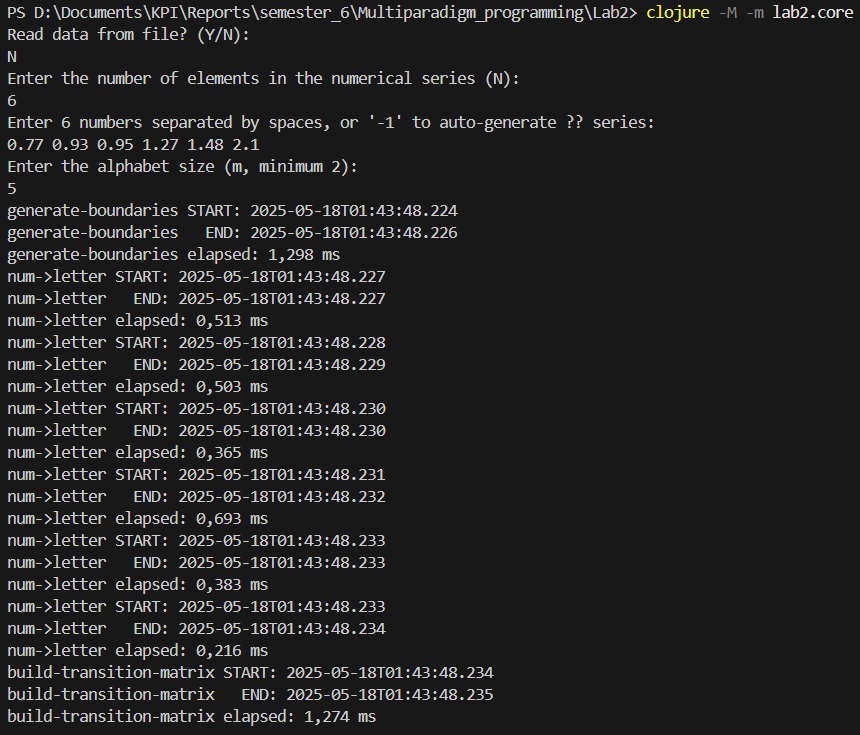


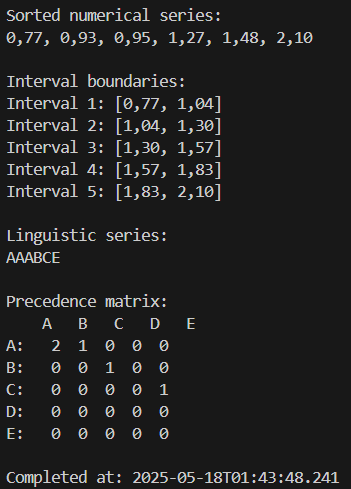
Генерація випадкових значень:





Введення значень від руки:

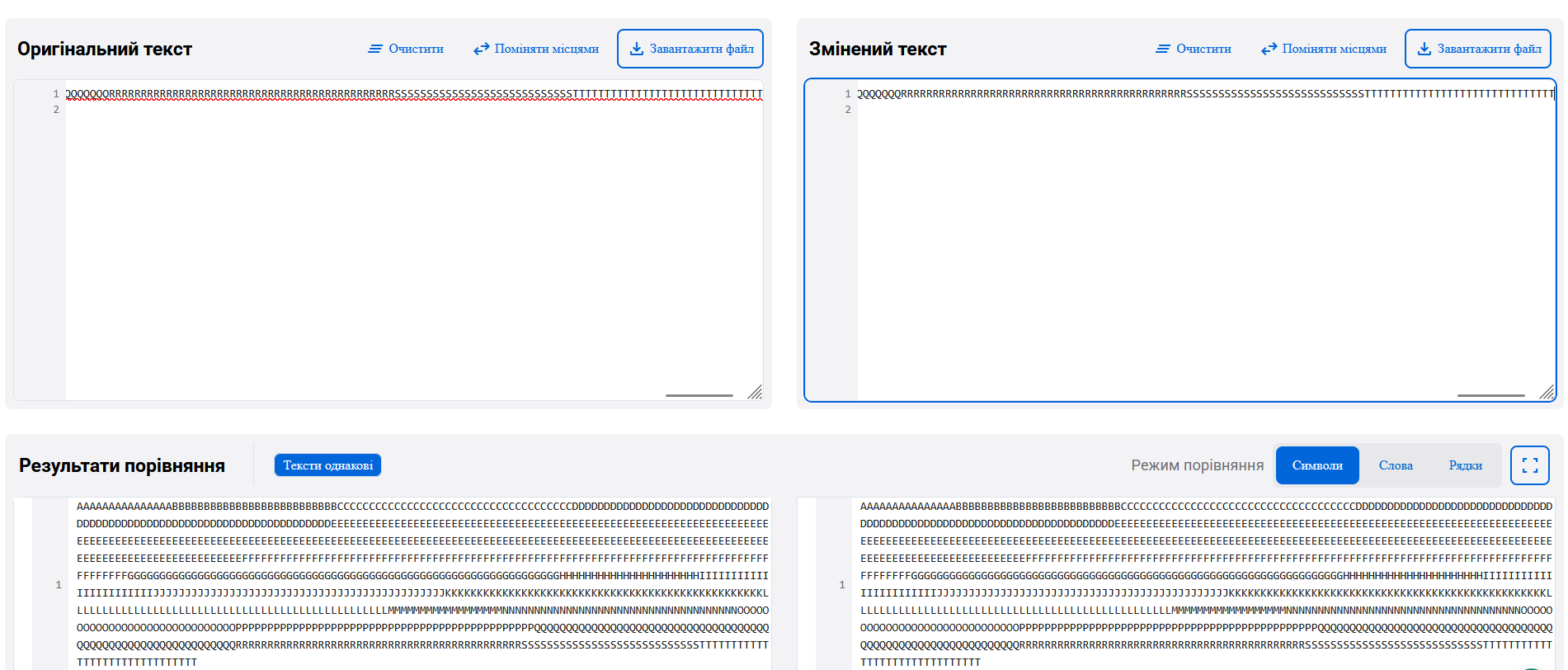




Поточний лінгвістичний ряд:

AAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGHHHHHHHHHHHHHHHHHHHHHHIIIIIIIIIIIIIIIIIIIIIIIJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLMMMMMMMMMMMMMMMMMMNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRSSSSSSSSSSSSSSSSSSSSSSSSSSSSTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

Порівняння лінгвістичних рядів лабораторної роботи 1 лабораторної роботи 2:

****

**Лінгвістичні ряди ідентичні.**

**Висновок**

У рамках лабораторної роботи №2 реалізовано функціональну програму на Clojure, яка:

* зчитує з консолі розмір числового ряду (N), потужність алфавіту (m) та сам ряд (або генерує його за χ²-розподілом);
* сортує вхідний числовий ряд у порядку зростання та визначає мінімальне й максимальне значення;
* обчислює межі m рівномірних за ймовірністю інтервалів через апроксимацію χ²-квантілей і виводить їх із часовими мітками й тривалістю розрахунку;
* перетворює кожне число на відповідну літеру алфавіту й формує лінгвістичний рядок із точною фіксацією часу виконання функції num→letter;
* будує матрицю передування символів із підрахунком частот переходів і виводить її разом із витраченим часом на обчислення.

Результати запусків підтверджують, що програма коректно працює для різних варіантів введених і згенерованих даних: відсортовані значення відповідають очікуванням, інтервали розбиті за рівними ймовірностями, лінгвістичний рядок відображає послідовність символів, а матриця передування чітко демонструє частоти переходів. Часові мітки й виміри швидкодії кожної ключової функції свідчать про ефективність реалізації. Таким чином, завдання ЛР №2 виконане повністю й відповідає вимогам ​

**Додатки**

Код програми:

(ns lab2.core

  (:gen-class)

  (:require [clojure.string :as str])

  (:import [java.time LocalDateTime]

           [java.time.format DateTimeFormatter]))

;; ==============================

;; 1. Ініціалізація форматтера часу

;; ==============================

(def ^DateTimeFormatter dtf

  (DateTimeFormatter/ofPattern "yyyy-MM-dd'T'HH:mm:ss.SSS"))

(defn now-str

  "Повертає поточну дату/час як відформатований рядок."

  []

  (.format (LocalDateTime/now) dtf))

;; ============================================

;; 2. Універсальна обгортка для заміру часу виконання

;; ============================================

(defn timed

  "Обгортка: виводить початок, кінець та тривалість виконання обчислення."

  [label body-fn]

  (let [start-time (LocalDateTime/now)

        start-ns   (System/nanoTime)]

    (println label "START:" (.format start-time dtf))

    (flush)

    (let [result (body-fn)

          end-time (LocalDateTime/now)

          elapsed (/ (- (System/nanoTime) start-ns) 1e6)]

      (println label "  END:" (.format end-time dtf))

      (println (format "%s elapsed: %.3f ms" label elapsed))

      (flush)

      result)))

;; ============================================

;; 3. Генерація рівномірних меж інтервалів

;; ============================================

(defn generate-boundaries

  "Генерує межі m рівних інтервалів між min-val і max-val."

  [min-val max-val m]

  (timed

   "generate-boundaries"

   #(vec (for [i (range 0 (inc m))]

           (+ min-val (\* i (/ (- max-val min-val) m)))))))

;; ====================================================

;; 4. Перетворення числа у літеру за межами інтервалів

;; ====================================================

(defn num->letter

  "Повертає літеру з alphabet, яка відповідає числу x за заданими межами."

  [x boundaries letters]

  (timed

   "num->letter"

   #(let [internal (subvec boundaries 1 (dec (count boundaries)))

          idx (loop [j 0]

                (if (or (= j (count internal)) (< x (nth internal j)))

                  j (recur (inc j))))]

      (nth letters idx))))

;; ============================================

;; 5. Побудова матриці передування (частот переходів)

;; ============================================

(defn build-transition-matrix

  "Будує матрицю частот переходів між літерами у letter-seq."

  [letter-seq alphabet]

  (timed

   "build-transition-matrix"

   #(let [m-count    (count alphabet)

          letter-map (zipmap alphabet (range m-count))

          empty-mat  (vec (repeat m-count (vec (repeat m-count 0))))

          transitions (partition 2 1 letter-seq)]

      (reduce (fn [mat [a b]]

                (update-in mat [(letter-map a) (letter-map b)] inc))

              empty-mat transitions))))

;; ===================================================

;; 6. Генерація випадкового χ²-розподілу

;; ===================================================

(defn generate-chi-square-series

  "Генерує N випадкових значень χ² з dof ступенями свободи."

  [N dof]

  (let [randgen (java.util.Random.)]

    (doall (repeatedly N

                       #(apply + (repeatedly dof

                                             (fn []

                                               (let [x (.nextGaussian randgen)]

                                                 (\* x x)))))))))

;; ===================================================

;; 7. Перетворення чисел у лінгвістичний ряд

;; ===================================================

(defn series->letters

  "Перетворює числову серію у відповідний ряд літер згідно меж."

  [series boundaries]

  (let [m       (dec (count boundaries))

        letters (vec (map char (range 65 (+ 65 m))))]

    (mapv #(num->letter % boundaries letters) series)))

;; ===================================================

;; 8. Зчитування з файлу стовпця Price

;; ===================================================

(defn read-price-column

  "Зчитує n значень з колонки 'Price' CSV-файлу filepath."

  [filepath n]

  (let [lines (str/split-lines (slurp filepath))

        headers (str/split (first lines) #",")

        price-index (.indexOf headers "Price")]

    (if (= price-index -1)

      (throw (Exception. "Column 'Price' not found in CSV."))

      (->> (rest lines)

           (map #(nth (str/split % #",") price-index))

           (map #(Double/parseDouble %))

           (take n)

           vec))))

;; ===================================================

;; 9. Головна функція виконання

;; ===================================================

(defn -main

  "Головна функція: зчитує вхідні дані, генерує інтервали, будує лінгвістичний ряд та матрицю переходів."

  [& args]

  ;; 9.1. Запит про джерело даних

  (println "Read data from file? (Y/N):")

  (flush)

  (let [ans (read-line)

        use-file (or (= ans "Y") (= ans "y"))

        ;; 9.2. Зчитування серії чисел

        series (if use-file

                 (do

                   (println "Enter how many numbers to read from data.csv:")

                   (flush)

                   (let [n (Integer/parseInt (read-line))]

                     (read-price-column "data.csv" n)))

                 (do

                   (println "Enter the number of elements in the numerical series (N):")

                   (flush)

                   (let [N (Integer/parseInt (read-line))]

                     (println (format "Enter %d numbers separated by spaces, or '-1' to auto-generate χ² series:" N))

                     (flush)

                     (let [line (read-line)]

                       (if (= line "-1")

                         (generate-chi-square-series N 4) ;; χ² з 4 ступенями свободи (варіант 4)

                         (mapv #(Double/parseDouble %) (str/split line #"\s+")))))))]

    ;; 9.3. Запит на розмір алфавіту

    (println "Enter the alphabet size (m, minimum 2):")

    (flush)

    (let [m (Integer/parseInt (read-line))

          sorted (vec (sort series))

          min-v  (first sorted)

          max-v  (last sorted)

          bounds (generate-boundaries min-v max-v m)

          letters (vec (map char (range 65 (+ 65 m))))

          letter-seq (series->letters sorted bounds)

          matrix (build-transition-matrix letter-seq letters)]

      ;; 10. Вивід результатів

      (println "\nSorted numerical series:")

      (println (str/join ", " (map #(format "%.2f" %) sorted)))

      (println "\nInterval boundaries:")

      (doseq [i (range 1 (inc m))]

        (let [l (nth bounds (dec i))

              u (nth bounds i)]

          (println (format "Interval %d: [%.2f, %.2f]" i l u))))

      (println "\nLinguistic series:")

      (println (apply str letter-seq))

      (println "\nPrecedence matrix:")

      (println (apply str "    " (interpose "   " (map str letters))))

      (doseq [i (range m)]

        (println (format "%c: %s"

                         (nth letters i)

                         (apply str (map #(format "%3d" %) (nth matrix i))))))

      ;; 11. Завершення з міткою часу

      (println "\nCompleted at:" (now-str)))))