Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 2

з дисципліни

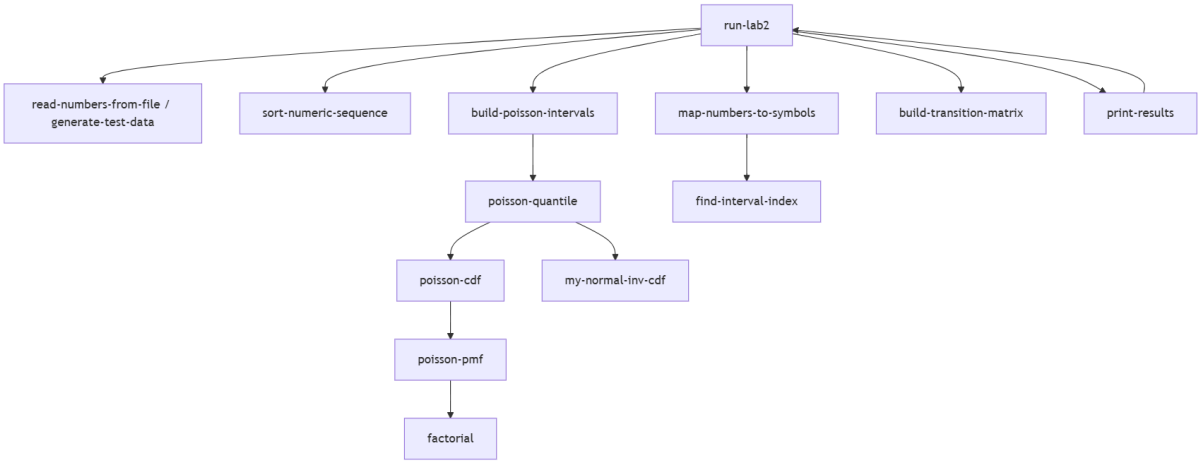
«Мультипарадигменне програмування»

ВИКОНАВ:
студент III курсу ФІОТ
групи ІО-23
Бичкар Н. В.
Залікова № 2302

ПЕРЕВІРИВ:
ас. Очеретяний О. К.

Київ – 2025

Функціональна схема (взаємозв'язку функцій):



Тексти визначень функцій:
Основні функції програми
1. Функції для генерації і зчитування даних
read-numbers-from-file

```
;; Зчитування чисел із текстового файлу
(define (read-numbers-from-file path)
  (with-handlers ([exn:fail? (lambda (e) '())])  ; Обробка помилок, якщо файл не існує
    (define content (file->string path))
    (define parts (string-split content))
    (map string->number parts)))
```

Призначення: Зчитує числові дані з текстового файлу, розділених пробілами.
generate-test-data

```
;; Генерація тестових даних (для уникнення залежності від зовнішнього файлу)
(define (generate-test-data n)
  (for/list ([i (in-range n)])
    (* 10 (random))))
```

Призначення: Генерує список випадкових чисел для тестування, якщо зовнішній файл недоступний.
2. Функції для обробки числових рядів
sort-numeric-sequence

```
;; Сортування числового ряду
(define (sort-numeric-sequence seq)
  (sort seq <))
```

Призначення: Сортує числовий ряд у порядку зростання.

factorial

```
;; Обчислення факторіалу (ітеративно для уникнення переповнення стеку)
(define (factorial n)
  (let loop ([i 1] [result 1])
    (if (> i n)
        result
        (loop (+ i 1) (* result i)))))
```

Призначення: Обчислює факторіал числа ітеративним методом для запобігання переповненню стеку.

poisson-pmf

```
;; Функція густини розподілу Пуассона P(X = k)
(define (poisson-pmf k lambda)
  (/ (* (expt lambda k) (exp (- lambda)))
     (factorial k)))
```

Призначення: Обчислює функцію густини ймовірності розподілу Пуассона P(X = k).

poisson-cdf

```
;; Функція розподілу Пуассона (кумулятивна) P(X <= k)
(define (poisson-cdf k lambda)
  (if (< k 0)
      0
      (let loop ([i 0] [sum 0])
        (if (> i k)
            sum
            (loop (+ i 1) (+ sum (poisson-pmf i lambda)))))))
```

Призначення: Обчислює кумулятивну функцію розподілу Пуассона P(X ≤ k).

my-normal-inv-cdf

```
;; Власна реалізація функції для обчислення квантиля нормального розподілу
(define (my-normal-inv-cdf prob mean stddev)
  ;; Використовуємо інверсію стандартного нормального розподілу
  ;; та лінійно трансформуємо її до потрібного mean і stddev
  (+ mean (* stddev (flnormal-inv-cdf (exact->inexact prob)))))
```

Призначення: Обчислює квантиль нормального розподілу з параметрами mean та stddev.

poisson-quantile

```
;; Функція для наближеного обчислення квантиля розподілу Пуассона
(define (poisson-quantile prob lambda)
  ;; Для великих значень lambda використовуємо нормальне наближення
  (if (> lambda 20)
      (exact-floor (+ (my-normal-inv-cdf prob lambda (sqrt lambda)) 0.5))
      ;; Інакше шукаємо бінарним пошуком з обмеженою кількістю ітерацій
      (let loop ([left 0] [right (max 100 (* 3 lambda))] [iteration 0])
        (if (or (<= (- right left) 0.5) (> iteration 50))  ; додаємо обмеження на ітерації
            (exact-ceiling left)
            (let* ([mid (exact-floor (/ (+ left right) 2))]
```

```
            [cdf-mid (poisson-cdf mid lambda)])
        (cond
          [(< (abs (- cdf-mid prob)) 0.0001) mid]
          [(< cdf-mid prob) (loop mid right (+ iteration 1))]
          [else (loop left mid (+ iteration 1))])))))
```

Призначення: Обчислює квантиль розподілу Пуассона, використовуючи різні підходи залежно від значення параметра lambda.

build-poisson-intervals

```
;; Побудова інтервалів за Пуассонівським розподілом
(define (build-poisson-intervals sorted-seq alphabet lambda)
  (define min-val (first sorted-seq))
  (define max-val (last sorted-seq))
  (define range (- max-val min-val))
  (define alphabet-size (length alphabet))

  ;; Обчислення меж інтервалів на основі квантилів Пуассона
  ;; Розбиваємо діапазон [0, 1] на рівні частини за ймовірністю
  (define prob-step (/ 1.0 alphabet-size))

  ;; Обчислюємо квантилі для кожної ймовірності
  (define quantiles
    (for/list ([i (in-range (+ alphabet-size 1))])
      (if (= i 0)
          0  ; Перший квантиль завжди 0
          (if (= i alphabet-size)
              (* 5 lambda)  ; Останній квантиль - достатньо велике число
              (poisson-quantile (* i prob-step) lambda)))))

  ;; Перетворюємо квантилі в межі інтервалів в оригінальному масштабі
  (define max-quantile (last quantiles))
  (define intervals
    (for/list ([i (in-range alphabet-size)])
      (list
       (+ min-val (* (/ (list-ref quantiles i) max-quantile) range))
       (+ min-val (* (/ (list-ref quantiles (+ i 1)) max-quantile) range)))))

  intervals)
```

Призначення: Будує інтервали для відображення числових значень на символи, використовуючи квантилі розподілу Пуассона.

find-interval-index

```
;; Знаходження індексу інтервалу, в який потрапляє значення
(define (find-interval-index value intervals)
  (let loop ([i 0] [ints intervals])
    (cond
      [(null? ints) (- (length intervals) 1)]
      [(and (<= (first (first ints)) value) (< value (second (first ints)))) i]
```

```
    ;; Додаткова перевірка на крайній правий кінець
    [(and (= i (- (length intervals) 1)) (<= value (second (first ints)))) i]
    [else (loop (+ i 1) (rest ints))])))
```

Призначення: Визначає, в який інтервал потрапляє задане числове значення.
map-numbers-to-symbols

```
;; Відображення чисел на символи алфавіту
(define (map-numbers-to-symbols seq intervals alphabet)
  (map (lambda (x) (list-ref alphabet (find-interval-index x intervals))) seq))
```

Призначення: Відображає кожне число з послідовності на відповідний символ алфавіту згідно з визначеними інтервалами.
3. Функції для побудови і виведення матриці передування
build-transition-matrix

```
;; Побудова матриці передування
(define (build-transition-matrix symbol-seq alphabet)
  (define size (length alphabet))
  ;; Створюємо матрицю як вектор векторів, заповнений нулями
  (define table (build-vector size (lambda (_) (make-vector size 0))))

  ;; Проходимо по всіх парах символів у ряді
  (for ([i (in-range (- (length symbol-seq) 1))])
    (let* ([a (index-of alphabet (list-ref symbol-seq i))]
           [b (index-of alphabet (list-ref symbol-seq (+ i 1)))])
      (vector-set! (vector-ref table a) b (+ 1 (vector-ref (vector-ref table a) b)))))

  table)
```

Призначення: Будує матрицю передування, яка показує частоту переходів між символами в послідовності.
print-results

```
;; Функція для виводу результатів
(define (print-results numeric-seq symbols intervals matrix alphabet lambda)
  ;; Виводимо опис завдання
  (displayln "============================================================")
  (displayln "Лабораторна робота №2 - Варіант 2. Пуассонівський розподіл")
  (displayln "============================================================")

  ;; Виводимо параметри
  (displayln (format "Розмір алфавіту: ~a" (length alphabet)))
  (displayln (format "Кількість чисел: ~a" (length numeric-seq)))
  (displayln (format "Параметр lambda: ~a" lambda))
  (displayln (format "Діапазон значень: від ~a до ~a"
                     (apply min numeric-seq)
                     (apply max numeric-seq)))

  ;; Виводимо перші декілька чисел
  (displayln "\nПерші 10 чисел вхідного ряду:")
```

```
(for ([i (in-range (min 10 (length numeric-seq)))])
  (display (format "~a " (list-ref numeric-seq i))))
(newline)

;; Виводимо інтервали
(displayln "\nІнтервали за Пуассонівським розподілом:")
(for ([i (in-range (length intervals))]
      [sym alphabet])
  (displayln (format "Інтервал ~a (~a): [~a, ~a)"
                     i sym
                     (real->decimal-string (first (list-ref intervals i)) 4)
                     (real->decimal-string (second (list-ref intervals i)) 4))))

;; Виводимо лінгвістичний ряд
(displayln "\nЛінгвістичний ряд (перші 50 символів):")
(for ([i (in-range (min 50 (length symbols)))])
  (display (list-ref symbols i)))
(newline)

;; Підраховуємо частоту символів
(displayln "\nЧастота символів:")
(for ([sym alphabet])
  (define count 0)
  (for ([s symbols])
    (when (equal? s sym)
      (set! count (+ count 1))))
  (displayln (format "~a: ~a (~a%)"
                     sym count
                     (real->decimal-string (* 100.0 (/ count (length symbols))) 2))))

;; Виводимо матрицю передування
(displayln "\nМатриця передування:")
;; Виводимо заголовки стовпців
(display "   ")
(for ([column alphabet])
  (display (format "~a  " column)))
(newline)

;; Виводимо роздільник
(display "  ")
(display (make-string (* 3 (length alphabet)) #\-))
(newline)

;; Виводимо кожен рядок з матриці
(for ([i (in-range (length alphabet))])
  (display (format "~a | " (list-ref alphabet i)))
  (for ([j (in-range (length alphabet))])
    (define val (vector-ref (vector-ref matrix i) j))
```

```
    (display (format "~a  " val)))
  (newline)))
```

Призначення: Виводить результати обробки даних, включаючи параметри, інтервали, лінгвістичний ряд та матрицю передування.

4. Основна функція виконання

run-lab2

```
;; Основна функція виконання завдання
(define (run-lab2 numeric-seq alphabet)
  ;; Параметр lambda для Пуассонівського розподілу (використовуємо середнє значення)
  (define lambda (/ (apply + numeric-seq) (length numeric-seq)))

  ;; Нормалізуємо lambda, якщо потрібно
  (define adjusted-lambda (max 1.0 lambda))

  ;; Сортуємо послідовність для знаходження діапазону
  (define sorted (sort-numeric-sequence numeric-seq))

  ;; Будуємо інтервали за Пуассонівським розподілом
  (define intervals (build-poisson-intervals sorted alphabet adjusted-lambda))

  ;; Відображаємо числа на символи
  (define symbols (map-numbers-to-symbols numeric-seq intervals alphabet))

  ;; Будуємо матрицю передування
  (define matrix (build-transition-matrix symbols alphabet))

  ;; Виводимо результати
  (print-results numeric-seq symbols intervals matrix alphabet adjusted-lambda)

  ;; Повертаємо результати для подальшого використання
  (values symbols matrix intervals))
```

Призначення: Координує виконання всіх етапів обробки - від підготовки даних до виведення результатів, є точкою входу в програму.

Лістинг:

```
#lang racket

;;
;; Лабораторна робота №2
;; Варіант 2. Перетворення чисельного ряду за Пуассонівським розподілом
```

```
;;
================================================================================
======

;; Підключаємо необхідні бібліотеки
(require math/statistics)
(require math/distributions)
;; Додатково імпортуємо функції для роботи з нормальним розподілом
(require math/special-functions)

;;
================================================================================
======
;; Допоміжні функції для генерації і зчитування даних
;;
================================================================================
======

;; Зчитування чисел із текстового файлу
(define (read-numbers-from-file path)
  (with-handlers ([exn:fail? (lambda (e) '())])  ; Обробка помилок, якщо файл не існує
    (define content (file->string path))
    (define parts (string-split content))
    (map string->number parts)))

;; Генерація тестових даних (для уникнення залежності від зовнішнього файлу)
(define (generate-test-data n)
  (for/list ([i (in-range n)])
    (* 10 (random))))

;;
================================================================================
======
;; Функції для обробки числових рядів
;;
================================================================================
======

;; Сортування числового ряду
(define (sort-numeric-sequence seq)
  (sort seq <))

;; Обчислення факторіалу (ітеративно для уникнення переповнення стеку)
(define (factorial n)
  (let loop ([i 1] [result 1])
    (if (> i n)
        result
        (loop (+ i 1) (* result i)))))
```

```
;; Функція густини розподілу Пуассона P(X = k)
(define (poisson-pmf k lambda)
  (/ (* (expt lambda k) (exp (- lambda)))
     (factorial k)))

;; Функція розподілу Пуассона (кумулятивна) P(X <= k)
(define (poisson-cdf k lambda)
  (if (< k 0)
      0
      (let loop ([i 0] [sum 0])
        (if (> i k)
            sum
            (loop (+ i 1) (+ sum (poisson-pmf i lambda)))))))

;; Власна реалізація функції для обчислення квантиля нормального розподілу
;; (заміна для normal-inv-cdf)
(define (my-normal-inv-cdf prob mean stddev)
  ;; Використовуємо інверсію стандартного нормального розподілу
  ;; та лінійно трансформуємо її до потрібного mean і stddev
  (+ mean (* stddev (flnormal-inv-cdf (exact->inexact prob)))))

;; Функція для наближеного обчислення квантиля розподілу Пуассона
;; (знаходження такого значення x, що CDF(x) = prob)
(define (poisson-quantile prob lambda)
  ;; Для великих значень lambda використовуємо нормальне наближення
  (if (> lambda 20)
      (exact-floor (+ (my-normal-inv-cdf prob lambda (sqrt lambda)) 0.5))
      ;; Інакше шукаємо бінарним пошуком з обмеженою кількістю ітерацій
      (let loop ([left 0] [right (max 100 (* 3 lambda))] [iteration 0])
        (if (or (<= (- right left) 0.5) (> iteration 50))  ; додаємо обмеження на ітерації
            (exact-ceiling left)
            (let* ([mid (exact-floor (/ (+ left right) 2))]
                   [cdf-mid (poisson-cdf mid lambda)])
              (cond
                [(< (abs (- cdf-mid prob)) 0.0001) mid]
                [(< cdf-mid prob) (loop mid right (+ iteration 1))]
                [else (loop left mid (+ iteration 1))])))))))

;; Побудова інтервалів за Пуассонівським розподілом
(define (build-poisson-intervals sorted-seq alphabet lambda)
  (define min-val (first sorted-seq))
  (define max-val (last sorted-seq))
  (define range (- max-val min-val))
  (define alphabet-size (length alphabet))

  ;; Обчислення меж інтервалів на основі квантилів Пуассона
  ;; Розбиваємо діапазон [0, 1] на рівні частини за ймовірністю
```

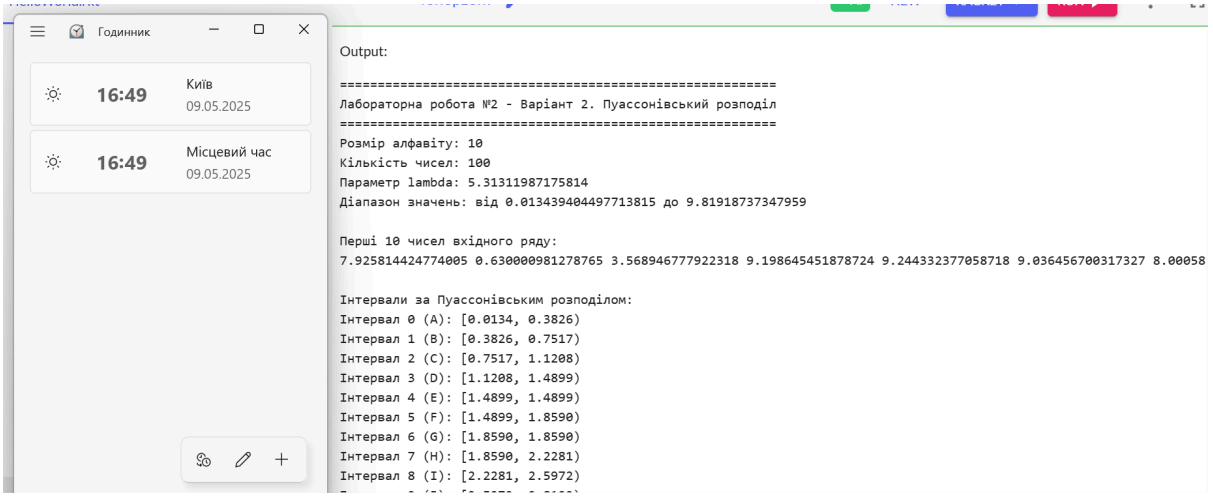
```
  (define prob-step (/ 1.0 alphabet-size))

  ;; Обчислюємо квантилі для кожної ймовірності
  (define quantiles
    (for/list ([i (in-range (+ alphabet-size 1))])
      (if (= i 0)
          0  ; Перший квантиль завжди 0
          (if (= i alphabet-size)
              (* 5 lambda)  ; Останній квантиль - достатньо велике число
              (poisson-quantile (* i prob-step) lambda)))))

  ;; Перетворюємо квантилі в межі інтервалів в оригінальному масштабі
  (define max-quantile (last quantiles))
  (define intervals
    (for/list ([i (in-range alphabet-size)])
      (list
       (+ min-val (* (/ (list-ref quantiles i) max-quantile) range))
       (+ min-val (* (/ (list-ref quantiles (+ i 1)) max-quantile) range)))))

  intervals)

;; Знаходження індексу інтервалу, в який потрапляє значення
(define (find-interval-index value intervals)
  (let loop ([i 0] [ints intervals])
    (cond
      [(null? ints) (- (length intervals) 1)]
      [(and (<= (first (first ints)) value) (< value (second (first ints)))) i]
      ;; Додаткова перевірка на крайній правий кінець
      [(and (= i (- (length intervals) 1)) (<= value (second (first ints)))) i]
      [else (loop (+ i 1) (rest ints))])))

;; Відображення чисел на символи алфавіту
(define (map-numbers-to-symbols seq intervals alphabet)
  (map (lambda (x) (list-ref alphabet (find-interval-index x intervals))) seq))

;;
================================================================================
;; Функції для побудови і виведення матриці передування
;;
================================================================================

;; Побудова матриці передування
(define (build-transition-matrix symbol-seq alphabet)
  (define size (length alphabet))
  ;; Створюємо матрицю як вектор векторів, заповнений нулями
  (define table (build-vector size (lambda (_) (make-vector size 0))))
```

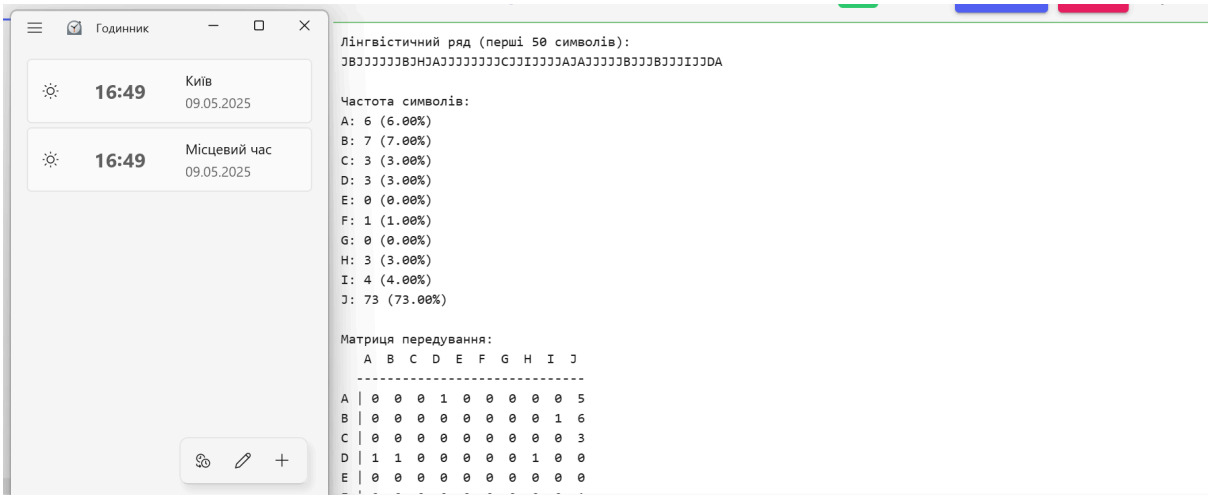
```
  ;; Проходимо по всіх парах символів у ряді
  (for ([i (in-range (- (length symbol-seq) 1))])
    (let* ([a (index-of alphabet (list-ref symbol-seq i))]
           [b (index-of alphabet (list-ref symbol-seq (+ i 1)))])
      (vector-set! (vector-ref table a) b (+ 1 (vector-ref (vector-ref table a) b)))))

  table)

;; Функція для виводу результатів
(define (print-results numeric-seq symbols intervals matrix alphabet lambda)
  ;; Виводимо опис завдання
  (displayln "============================================================")
  (displayln "Лабораторна робота №2 - Варіант 2. Пуассонівський розподіл")
  (displayln "============================================================")

  ;; Виводимо параметри
  (displayln (format "Розмір алфавіту: ~a" (length alphabet)))
  (displayln (format "Кількість чисел: ~a" (length numeric-seq)))
  (displayln (format "Параметр lambda: ~a" lambda))
  (displayln (format "Діапазон значень: від ~a до ~a"
                     (apply min numeric-seq)
                     (apply max numeric-seq)))

  ;; Виводимо перші декілька чисел
  (displayln "\nПерші 10 чисел вхідного ряду:")
  (for ([i (in-range (min 10 (length numeric-seq)))])
    (display (format "~a " (list-ref numeric-seq i))))
  (newline)

  ;; Виводимо інтервали
  (displayln "\nІнтервали за Пуассонівським розподілом:")
  (for ([i (in-range (length intervals))]
        [sym alphabet])
    (displayln (format "Інтервал ~a (~a): [~a, ~a)"
                       i sym
                       (real->decimal-string (first (list-ref intervals i)) 4)
                       (real->decimal-string (second (list-ref intervals i)) 4))))

  ;; Виводимо лінгвістичний ряд
  (displayln "\nЛінгвістичний ряд (перші 50 символів):")
  (for ([i (in-range (min 50 (length symbols)))])
    (display (list-ref symbols i)))
  (newline)

  ;; Підраховуємо частоту символів
  (displayln "\nЧастота символів:")
  (for ([sym alphabet])
```

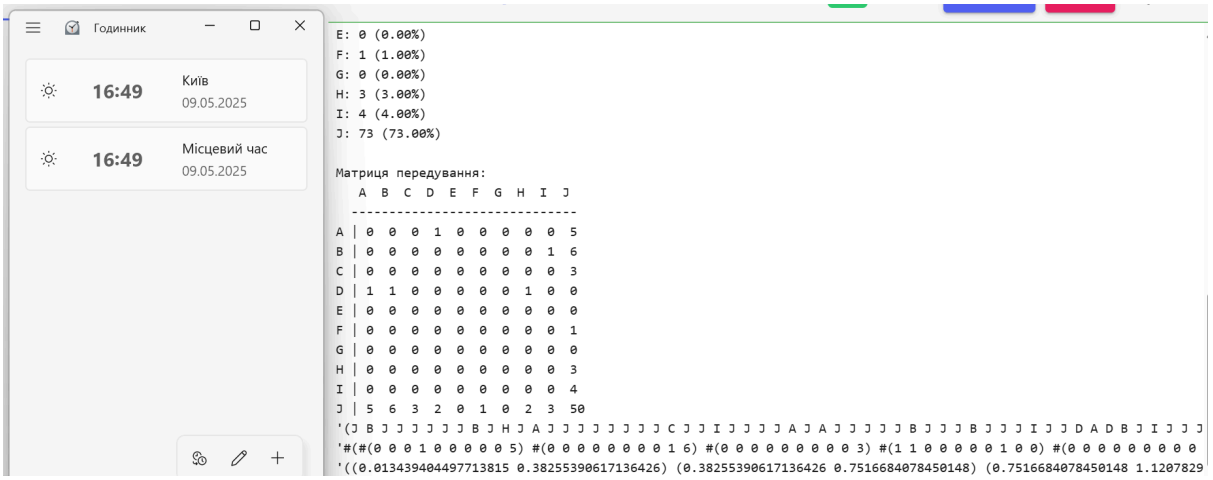
```
    (define count 0)
    (for ([s symbols])
      (when (equal? s sym)
        (set! count (+ count 1))))
    (displayln (format "~a: ~a (~a%)"
                       sym count
                       (real->decimal-string (* 100.0 (/ count (length symbols))) 2))))

  ;; Виводимо матрицю передування
  (displayln "\nМатриця передування:")
  ;; Виводимо заголовки стовпців
  (display "   ")
  (for ([column alphabet])
    (display (format "~a  " column)))
  (newline)

  ;; Виводимо роздільник
  (display "  ")
  (display (make-string (* 3 (length alphabet)) #\-))
  (newline)

  ;; Виводимо кожен рядок з матриці
  (for ([i (in-range (length alphabet))])
    (display (format "~a | " (list-ref alphabet i)))
    (for ([j (in-range (length alphabet))])
      (define val (vector-ref (vector-ref matrix i) j))
      (display (format "~a  " val)))
    (newline)))

;;
================================================================================
;; Основна функція виконання завдання
;;
================================================================================

(define (run-lab2 numeric-seq alphabet)
  ;; Параметр lambda для Пуассонівського розподілу (використовуємо середнє
значення)
  (define lambda (/ (apply + numeric-seq) (length numeric-seq)))

  ;; Нормалізуємо lambda, якщо потрібно
  (define adjusted-lambda (max 1.0 lambda))

  ;; Сортуємо послідовність для знаходження діапазону
  (define sorted (sort-numeric-sequence numeric-seq))
```

```
  ;; Будуємо інтервали за Пуассонівським розподілом
  (define intervals (build-poisson-intervals sorted alphabet adjusted-lambda))

  ;; Відображаємо числа на символи
  (define symbols (map-numbers-to-symbols numeric-seq intervals alphabet))

  ;; Будуємо матрицю передування
  (define matrix (build-transition-matrix symbols alphabet))

  ;; Виводимо результати
  (print-results numeric-seq symbols intervals matrix alphabet adjusted-lambda)

  ;; Повертаємо результати для подальшого використання
  (values symbols matrix intervals))

;;
================================================================================
;; Параметри запуску та виклик основної функції
;;
================================================================================

;; Генеруємо тестові дані
(define numeric-seq (generate-test-data 100))

;; Можна також зчитати дані з файлу
;; (define numeric-seq (read-numbers-from-file "data.txt"))

;; Альтернативно, можна задати конкретні значення для тестування
;; (define numeric-seq '(2.5 3.7 1.2 4.8 5.1 2.2 3.8 4.3 1.9 5.0
;;                       3.1 4.2 2.8 5.5 3.4 1.7 4.9 2.3 3.5 4.1
;;                       1.4 3.3 5.2 2.9 4.7 3.2 1.8 4.5 2.0 5.9))

;; Задаємо алфавіт
(define alphabet '(A B C D E F G H I J))

;; Запускаємо програму
(run-lab2 numeric-seq alphabet)
```

Результати:







==========
Лабораторна робота №2 - Варіант 2. Пуассонівський розподіл
==========
Розмір алфавіту: 10
Кількість чисел: 100
Параметр lambda: 5.31311987175814
Діапазон значень: від 0.013439404497713815 до 9.81918737347959

Перші 10 чисел вхідного ряду:

7.925814424774005 0.630000981278765 3.568946777922318 9.198645451878724
9.244332377058718 9.036456700317327 8.00058603848375 5.900347758380774
0.7060125625810152 8.537310845162873

Інтервали за Пуассонівським розподілом:
Інтервал 0 (A): [0.0134, 0.3826)
Інтервал 1 (B): [0.3826, 0.7517)
Інтервал 2 (C): [0.7517, 1.1208)
Інтервал 3 (D): [1.1208, 1.4899)
Інтервал 4 (E): [1.4899, 1.4899)
Інтервал 5 (F): [1.4899, 1.8590)
Інтервал 6 (G): [1.8590, 1.8590)
Інтервал 7 (H): [1.8590, 2.2281)
Інтервал 8 (I): [2.2281, 2.5972)
Інтервал 9 (J): [2.5972, 9.8192)

Лінгвістичний ряд (перші 50 символів):
JBJJJJJJBJHJAJJJJJJJJCJJIJJJJAJAJJJJJBJJJBJJJIJJDA

Частота символів:
A: 6 (6.00%)
B: 7 (7.00%)
C: 3 (3.00%)
D: 3 (3.00%)
E: 0 (0.00%)
F: 1 (1.00%)
G: 0 (0.00%)
H: 3 (3.00%)
I: 4 (4.00%)
J: 73 (73.00%)

Матриця передування:
```
   A  B  C  D  E  F  G  H  I  J
  ------------------------------
A | 0  0  0  1  0  0  0  0  0  5
B | 0  0  0  0  0  0  0  0  1  6
C | 0  0  0  0  0  0  0  0  0  3
D | 1  1  0  0  0  0  0  1  0  0
E | 0  0  0  0  0  0  0  0  0  0
F | 0  0  0  0  0  0  0  0  0  1
G | 0  0  0  0  0  0  0  0  0  0
H | 0  0  0  0  0  0  0  0  0  3
I | 0  0  0  0  0  0  0  0  0  4
J | 5  6  3  2  0  1  0  2  3  50
```
'(J B J J J J J J B J H J A J J J J J J J J C J J I J J J J A J A J J J J J B J J J B J J J I J J D A
D B J I J J J J J J J D H J J J J J J J J J B I J H J J A J B J C J J J J J J C J J J J J A J F J J)

'#(#(0 0 0 1 0 0 0 0 0 5) #(0 0 0 0 0 0 0 0 1 6) #(0 0 0 0 0 0 0 0 0 3) #(1 1 0 0 0 0 0 1 0 0) #(0 0 0 0 0 0 0 0 0 0) #(0 0 0 0 0 0 0 0 0 1) #(0 0 0 0 0 0 0 0 0 0) #(0 0 0 0 0 0 0 0 0 3) #(0 0 0 0 0 0 0 0 0 4) #(5 6 3 2 0 1 0 2 3 50))
'((0.013439404497713815 0.38255390617136426) (0.38255390617136426 0.7516684078450148) (0.7516684078450148 1.120782909518665) (1.120782909518665 1.4898974111923156) (1.4898974111923156 1.4898974111923156) (1.4898974111923156 1.859011912865966) (1.859011912865966 1.859011912865966) (1.859011912865966 2.2281264145396165) (2.2281264145396165 2.597240916213267) (2.597240916213267 9.81918737347959))