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 ІК-21

Бераудо Лоренцо Раффаелович

Київ 2025

ЛАБОРАТОРНА 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 1: Дискретний рівномірний розподіл (рівноймовірний)

ХІД ВИКОНАННЯ ЗАВДАННЯ

1. Генерація чисельного ряду

Для прикладу було згенеровано 100 випадкових чисел у діапазоні від 1 до 1000. Це виконано за допомогою функції generate-random-list:

```
(defn generate-random-list [size min-val max-val]
(repeatedly size #(+ min-val (rand-int (- max-val min-val)))))
```

2. Сортування ряду

Сортування необхідне для точного визначення діапазону значень, які будуть розбиті на інтервали.

```
(def sorted-list (sort num-list))
```

3. Розбиття на інтервали

Для рівномірного розподілу використовується функція uniform-segment-ends, яка ділить діапазон на рівні частини згідно з потужністю алфавіту:

```
(defn uniform-segment-ends [min-val max-val count]
(let [step (double (/ (- max-val min-val) count))]
(map #(int (+ min-val (* % step))) (range count))))
```

4. Перетворення чисел у символи алфавіту

Кожне число співвідноситься з відповідним символом, який відповідає інтервалу, у який потрапляє значення.

```
(defn numbers-to-letters [numbers segments alphabet]
  (map #(nth alphabet (find-segment-index % segments)) numbers))
```

5. Побудова матриці передування

Створюється асоціативна структура, в якій підраховується кількість пар переходів символів у лінгвістичному ряді:

```
(defn build-transition-matrix [letters alphabet]
  (let [matrix (atom (zipmap alphabet (repeat (zipmap alphabet (repeat 0)))))]
    (doseq [[a b] (partition 2 1 letters)]
      (swap! matrix update-in [a b] inc))
    @matrix))
```

6. Повна обробка

Основна функція process-sequence викликає всі етапи й виводить результат:

```
(defn process-sequence []
  (let [num-list (generate-random-list num-size min-val max-val)
        sorted-list (sort num-list)
        segments (uniform-segment-ends (first sorted-list) (last sorted-list) alphabet-power)
        letters (numbers-to-letters num-list segments alphabet)
        transition-matrix (build-transition-matrix letters alphabet)]
    (println "Числовий ряд:" num-list)
    (println "Лінгвістичний ряд:" (str/join " " letters))
    (println "Матриця передування:")
    (doseq [[key val] transition-matrix]
      (println key val))))
```

ПРИКЛАД ВИВОДУ

Числовий ряд:

```
Числовий ряд: (893 287 562 520 988 919 394 320 555 998 983 948 513 372 514 366 861 826
```

Лінгвістичний ряд:

```
Лінгвістичний ряд: i c f f j j d d f j j j f d f d i i e g c b j d h e h d i b d j a c
```

Матриця передування (фрагмент):

```
Матриця передування:
a {a 2, b 0, c 2, d 0, e 1, f 0, g 2, h 2, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
b {a 0, b 0, c 0, d 4, e 0, f 0, g 0, h 1, i 1, j 2, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
c {a 1, b 1, c 1, d 0, e 0, f 4, g 1, h 1, i 1, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
d {a 0, b 2, c 1, d 1, e 0, f 2, g 1, h 1, i 2, j 1, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
e {a 1, b 1, c 0, d 0, e 0, f 0, g 1, h 1, i 1, j 1, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
f {a 0, b 0, c 2, d 2, e 0, f 1, g 0, h 1, i 0, j 3, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
g {a 2, b 1, c 1, d 1, e 1, f 0, g 1, h 0, i 1, j 2, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
h {a 1, b 0, c 0, d 1, e 2, f 1, g 1, h 1, i 2, j 1, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
i {a 0, b 2, c 1, d 0, e 2, f 0, g 2, h 1, i 1, j 2, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
j {a 3, b 1, c 2, d 2, e 0, f 1, g 1, h 1, i 1, j 3, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
k {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
l {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
m {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
n {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
o {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
p {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
q {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
r {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
s {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
t {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
u {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
v {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
w {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
x {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
y {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
z {a 0, b 0, c 0, d 0, e 0, f 0, g 0, h 0, i 0, j 0, k 0, l 0, m 0, n 0, o 0, p 0, q 0, r 0, s 0, t 0, u 0, v 0, w 0, x 0, y 0, z 0}
```

ВИСНОВОК

У ході виконання лабораторної роботи було реалізовано алгоритм трансформації чисельного ряду у лінгвістичний ланцюжок відповідно до рівномірного розподілу.
Кожному числу призначено символ згідно з розбиттям на інтервали, а побудована матриця передування наочно демонструє частоту переходів між символами.

Робота дозволила отримати практичні навички:

- функціональної обробки масивів,
- побудови асоціативних структур,
- реалізації ймовірнісних алгоритмів,
- функціонального стилю програмування на мові **Clojure**.

Код програми

```
(ns lab2.core
  (:require [clojure.string :as str]))

;; === Вхідні дані ===
(def alphabet (vec "abcdefghijklmnopqrstuvwxyz")) ;; Алфавіт із 26
літер
(def alphabet-power 10) ;; Потужність алфавіту
(def num-size 100) ;; Кількість чисел у ряді
(def min-val 1)  ;; Мінімальне значення у ряді
(def max-val 1000) ;; Максимальне значення у ряді

;; === Генерація випадкового числового ряду ===
(defn generate-random-list [size min-val max-val]
  "Генерує випадковий числовий ряд із заданим розміром та межами"
  (repeatedly size #(+ min-val (rand-int (- max-val min-val)))))

;; === Розбиття числового ряду на рівномірні інтервали ===
(defn uniform-segment-ends [min-val max-val count]
  "Створює список кінцевих точок інтервалів рівномірного
розподілу"
  (let [step (double (/ (- max-val min-val) count))]
    (map #(int (+ min-val (* % step))) (range count))))

;; === Визначення індексу інтервалу для числа ===
(defn find-segment-index [num segments]
  "Знаходить індекс інтервалу, до якого належить число"
  (let [idx (->> segments
                 (map-indexed vector)
                 (filter #(<= (second %) num))
                 last
                 first)]
    (if (nil? idx) 0 idx)))

;; === Перетворення чисел у літери ===
(defn numbers-to-letters [numbers segments alphabet]
  "Перетворює числовий ряд у літери відповідно до розподілу
інтервалів"
  (map #(nth alphabet (find-segment-index % segments)) numbers))

;; === Побудова матриці передування ===
(defn build-transition-matrix [letters alphabet]
  "Будує матрицю передування для лінгвістичного ряду"
  (let [matrix (atom (zipmap alphabet (repeat (zipmap alphabet
(repeat 0)))))]
    (doseq [[a b] (partition 2 1 letters)]
      (swap! matrix update-in [a b] inc))
    @matrix))

;; === Основна функція виконання ===
(defn process-sequence []
  "Головна функція, що виконує всі етапи обробки даних"
```

```
  (let [num-list (generate-random-list num-size min-val max-val)
;; Генеруємо випадковий ряд
        sorted-list (sort num-list) ;; Сортуємо список
        segments (uniform-segment-ends (first sorted-list) (last
sorted-list) alphabet-power) ;; Розбиваємо на інтервали
        letters (numbers-to-letters num-list segments alphabet) ;;
Перетворюємо числа у літери
        transition-matrix (build-transition-matrix letters
alphabet)] ;; Будуємо матрицю передування
    ;; Виведення результатів
    (println "Числовий ряд:" num-list)
    (println "Лінгвістичний ряд:" (str/join " " letters))
    (println "Матриця передування:")
    (doseq [[key val] transition-matrix]
      (println key val))))

;; === Виконання програми ===
(process-sequence)
```