Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2

Мультипарадигменне програмування

Виконав
студент групи ІА-23:
Курач В.А.

Київ 2025

Завдання: на процедурній мові 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.

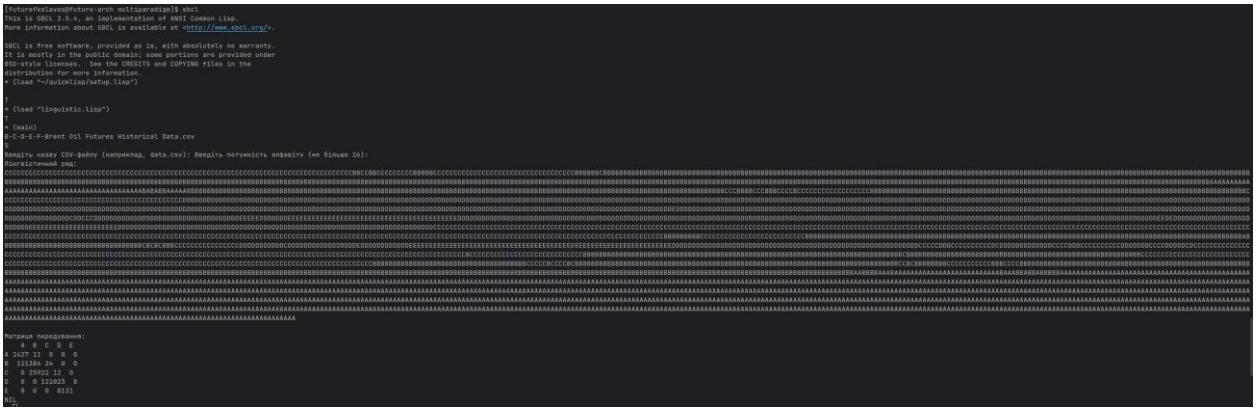
Вихідні дані: лінгвістичний ряд та матриця передування.

Хід розв’язання задачі:

1. Чисельний ряд сортується від найменшого значення до найбільшого, що дає діапазон припустимих значень.
2. Розподіл ймовірностей використовується для розбиття області значень на інтервали, кількість яких відповідає потужності обраного алфавіту.
3. Кожному числовому значенню ставиться у відповідність символ алфавіту, індекс якого відповідає інтервалу, в який потрапляє це значення.
4. Отриманий лінгвістичний ряд виводиться на вихід.
5. На основі лінгвістичного ряду будується матриця передування, де елементи відображають кількість переходів від однієї літери до іншої у ряді.

Використані бібліотеки:

- **parse-number** — для конвертації рядкових представлень чисел у числові значення з плаваючою точкою.
- Вбудовані стандартні функції Common Lisp для роботи з файлами, рядками, масивами, обробки помилок та сортування.



Лістинг програми:

```
(ql:quickload "parse-number" :silent t)

(defun parse-float (str)
  (handler-case (parse-number:parse-number str)
    (error () nil)))

(defun split-string (str &optional (delimiter #\,))
  (loop with start = 0
        for pos = (position delimiter str :start start)
        collect (subseq str start (or pos (length str)))
        while pos
        do (setf start (1+ pos))))

(defun extract-field (line field-num)
  (let ((fields (split-string line #\,)))
    (if (<= field-num (length fields))
        (string-trim "\" " (nth (1- field-num) fields))
        "")))

(defun read-csv-column-dynamic (filename column-index)
  (with-open-file (stream filename :direction :input)
    (ignore-errors (read-line stream))
    (let ((values '()))
      (loop
        for line = (ignore-errors (read-line stream nil nil))
        while line
        do (let* ((field (extract-field line (1+ column-index)))
                  (val (parse-float field)))
             (when val
               (push (coerce val 'double-float) values))))
      (let* ((values (nreverse values))
             (arr (make-array (length values) :element-type 'double-float)))
        (loop for i from 0 below (length values) do
             (setf (aref arr i) (nth i values)))
        arr))))

(defun sort-array (arr)
  (let ((copy (coerce arr 'list)))
    (sort copy #'<)))

(defun generate-alphabet (size)
  (loop for i from 0 below size
        collect (code-char (+ (char-code #\A) i))))

(defun index-in-alphabet (symbol alphabet)
  (position symbol alphabet))

(defun to-linguistic (series alphabet)
  (let* ((alpha-size (length alphabet))
         (min-val (reduce #'min series))
         (max-val (reduce #'max series))
         (interval (/ (- max-val min-val) alpha-size)))
    (map 'vector
         (lambda (val)
           (let ((index (min (floor (/ (- val min-val) interval)) (1- alpha-size))))
             (nth index alphabet)))
         series)))
```

```
(defun build-transition-matrix (ling-series alphabet)
  (let* ((size (length alphabet))
         (matrix (make-array (list size size) :initial-element 0)))
    (loop for i from 0 below (1- (length ling-series)) do
      (let ((curr (index-in-alphabet (aref ling-series i) alphabet))
            (next (index-in-alphabet (aref ling-series (1+ i)) alphabet)))
        (when (and curr next)
          (incf (aref matrix curr next)))))
    matrix))

(defun print-linguistic-series (ling-series)
  (format t "~%Лінгвістичний ряд:~%")
  (loop for ch across ling-series do (format t "~A" ch))
  (terpri))

(defun print-matrix (matrix alphabet)
  (format t "~%Матриця передування:~%    ")
  (dolist (ch alphabet)
    (format t "~3A" ch))
  (terpri)
  (loop for i from 0 below (length alphabet) do
      (format t "~A " (nth i alphabet))
      (loop for j from 0 below (length alphabet) do
          (format t "~3D" (aref matrix i j)))
      (terpri)))

(defun read-alpha-size ()
  (loop
    do (format t "Введіть потужність алфавіту (не більше 26): ")
      (let ((input (string-trim " \t\n\r" (read-line))))
        (let ((num (parse-integer input :junk-allowed t)))
          (if (and num (<= 1 num 26))
              (return num)
              (format t "Помилка: введіть ціле число від 1 до 26.~%"))))))

(defun main ()
  (format t "Введіть назву CSV-файлу (наприклад, data.csv): ")
  (let ((filename (string-trim " \t\n\r" (read-line))))
    (let ((series (read-csv-column-dynamic filename 1)))
      (if (or (null series) (= (length series) 0))
          (format t "Файл порожній або не містить значень.~%")
          (let* ((alpha-size (read-alpha-size))
                 (alphabet (generate-alphabet alpha-size))
                 (ling-series (to-linguistic series alphabet))
                 (matrix (build-transition-matrix ling-series alphabet)))
            (print-linguistic-series ling-series)
            (print-matrix matrix alphabet))))))
```

Висновки: У результаті виконання лабораторної роботи було успішно реалізовано перетворення числового ряду в лінгвістичний ланцюжок з урахуванням заданого розподілу ймовірностей та потужності алфавіту. Побудовано матрицю передування, яка відображає частоту переходів між символами і може бути використана для подальшого статистичного аналізу. Отримано практичні навички роботи з файлами, обробки числових даних,

маніпуляції масивами, сортуванням, а також реалізації функціональних процедур у мові Common Lisp.