# Міністерство освіти і науки України НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Лабораторна робота № 2 з дисципліни «Мультипарадигмене програмування»

> Виконав: Студент групи IO-23 Швед А. Д.

## Завдання

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.

# Хід роботи

Програма реалізує перетворення числового ряду у лінгвістичний ланцюжок з наступним побудуванням матриці передування. Алгоритм було реалізовано мовою Fortran стандарту F90. У імплементації використовуються динамічні масиви для роботи з великою кількістю даних у пам'яті та гнучкості програми.

Алгоритм можна поділити на 6 основних етапів:

- 1. Створення алфавіту довжини визначеної користувачем.
- 2. Читання числового ряду з файлу у список.
- 3. Сортування числового списку.
- 4. Побудова лінгвістичного ряду.
- 5. Побудова матриці передування.
- 6. Виведення результатів.

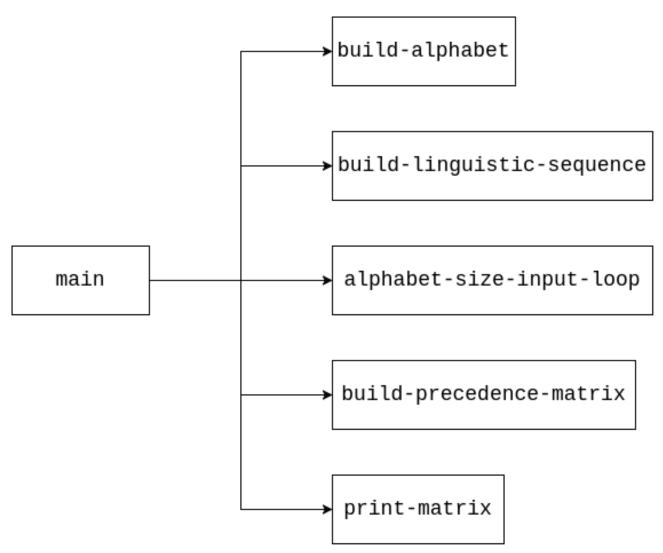


Рисунок 1: Функціональна схема

## Результати виконання

```
.
Enter alphabet size: 5
Alphabet Size: 5
Alphabet: ABCDE
Precefence Matrix:
12
1023
```

Рисунок 2: Перший числовий ряд B-C-D-E-F-Brent Oil Futures Historical Data (5000 значень - 5 символів)

and the second s	@rhinemanr		(/S/m/wor	kspace (	(main)> r	acket <u>la</u>	ab_2/mair	<u>n.rkt</u>						
Enter alp	habet size	e: 15												
Alphabet Size: 15														
Alphabet: ABCDEFGHIJKLMNO														
Linquistic Sequence: HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH														
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEDDDDDEEEE														
FFFFFFEEFFFFFFFFEFFEEFFFFEEFFFFFFFFFFEEEE														
EEEEEEEEDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD														
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC														
EEEEEDDDDEEEEEEEEEEEEEEEEEFFFFFFFFFFFFF														
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF														
JJJKKKJKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK														
KKKKKKKKKKLKLLLLLKKKKKKKKKKKKKKKKKKKKKK														
KLKLLLLLLLLLLLLLLLLLLLLLKKKKKKKKKKKKKKK														
KKKKKKKLKKKLLILLLLLLLLLLLLLLLLLLLLLLLLL														
LLLLLLKLLLLLLLLLLLLLLLLLLLLLLLLLLKKKKKK														
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM														
KKJKKKKKKKLLKLLLLLLLLLLLLLLLKKKKLKLKKKKKK														
MMMMMMLLLLLLLLLLLLLLLLLLLLLLLLLLLKKKKKKK														
ІНІІІІІІІІІІННИНИННИНИНИНИННИННИННИННИНН														
	GGGGGGGGG													
	EEEEEEEEE													
	GGGGGHHHII													
	LLLLLLLLL													
JJJJJIJII	IIIIIIIIII	ІІННННННІ	нниннин	ННННННН	нннннне	GGGGGGGG	GGGGGGGG	GGGGHHHHI	нннннннн	нининини	HGGGGGGG	GGGGGGGG	GGGGGGGG	BOBBOBBOB
GGGGGGGGG	FGGGGGGGG	6666666666	666666666	GGFFFFF	FFFFFFF	FFFFFFF	FFFFFFF	FFFFFFFF	FFFEEEFE	EEEEEEEE	EFFEFFFF	FFFFFFF	FFFFFFFF	GFFFFFFF
FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFF	FFFFFFF	FFFFFFGG	eeeeeeee	eeeeeee	<mark>веннннн</mark>	ннннннын	ндиннинн	нннееннее	GGGGGGG	GGGGGGGG	999999999
	<b>GGGGGGGGH</b>													
FFFFFFFF	EEFFFFFFF	FEFFFFFF	FEFFFFF	FFFFFFF	FEGGEGE	FFFFFF	าลลลลลลล	GGEFEGGG	GEFEFFF	FEFFFFFF	FFFFFFFF	FFFFFFF	FEFFFFFF	FFFFFFFF
	EEEEEEEEE													
	DDDDDDDEDEE													
	DCDDDDDCCC													
	CCCCCCCCE													
	BBBBBBBBB													
	BBBBBBBBB													
	BBBBBBBBB													
AAAABBAAA	AAABBBBBAE	BBBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BCCBBCBE	BBBBBBBB	BBBBBBBB	BBBBBBBBB	BBBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBBB	BBCCCCCCC
CCCCCCCC	CCCCCBCCCE	BCBBCBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BCCCCCCC	CBBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBCC
CCCCCCCC	cccccccc	cccccccc	ccccccc	ccccccc	ccccccc	ccccccc	ccccccc	CCCCCCC	CCCCCCCC	CCBBBBBBB	BBCCCCCC	CCCCCCC	cccccccc	BCCCCCCC
cccccccc	ВВССВССВВЕ	вввввввв	вввввввв	ВВВВВВВВ	ВВВВВВВВ	ввввессс	ССССССВЕ	вввввввв	ввсввввв	вввввввв	вввввввв	BBBBBBBB	ВВВВВВВВЕ	BBBBBBBBB
вввввввв	ВВВВВВВВВ	ВВВВВВВВВВ	ВВВВВВВВ	BBBBBBB	ВВВВВВВВ	ВВВВВВВВ	ВВВВВВВВ	вввввввв	вввввввв	ВВВВВВВВВ	BAAABBAAA	AAAAAAA	<b>AAAAAAA</b>	AAAAAAAA
	AAAAAAAAA													
Precefenc											,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	· · · · · · · · · · · · · · · · · · ·		
A		С	D	Е	F	G	н	I	J	К	L	М	N	0
A 154		0	0	0	0	0	п 0	0	0	0	0	0	N 0	0
III														
В 8		29	0	0	0	0	0	0	0	0	0	0	0	0
C 0		489	11	0	0	0	0	0	0	0	0	0	0	0
D 0		12	269	26	0	0	0	0	0	0	0	0	0	0
E 0	0	0	27	511	29	0	0	0	0	0	0	0	0	0
F 0	0	0	0	30	492	24	0	0	0	0	0	0	0	0
G 0	0	0	0	0	25	404	26	0	0	0	0	0	0	Θ
н о		0	0	0	0	27	311	7	0	0	0	0	0	0
i o		0	0	0	0	0	7	140	12	0	0	0	0	0
ن ا		0	0	0	0	0	0	12	136	13	0	0	0	0
K 0		0	0	0	0	0	0	0	138	393	46	0	0	0
III														
L 0		0	0	0	0	0	0	0	0	46	376	8	0	0
M 0		0	0	0	0	0	0	0	0	0	8	88	3	0
N 0		0	0	0	0	0	0	0	0	0	0	2	22	3
0 0	0	0	0	0	0	0	0	0	0	0	0	1	2	10

Рисунок 3: Перший числовий ряд B-C-D-E-F-Brent Oil Futures Historical Data (5000 значень - 15 символів)

# Лістинг коду

```
#lang racket
    (require racket/format)
   ; Alphabet builder function
   (define (build-alphabet size)
      (map (lambda (i) (string (integer->char (+ 65 i)))) (range size)))
10 : Linguistic sequence builder function
11 (define (build-linguistic-sequence numbers sorted alphabet)
      (define min_number (first sorted))
12
13
      (define max_number (last sorted))
      (define size (length alphabet))
14
15
      (define interval (/ (- max_number min_number) size))
16
      (map (lambda (num)
             (define idx (inexact->exact (ceiling (/ (- num min_number) interval))))
18
             (list-ref alphabet (cond [(< idx 1) 0]
                                       [(>= idx size) (sub1 size)]
20
                                       [else (sub1 idx)])))
21
           numbers))
23
    ; Build precedence matrix (vector of vectors)
24
    (define (build-precedence-matrix sequence alphabet)
26
      (define size (length alphabet))
      (define matrix (build-vector size (lambda (_) (make-vector size 0))))
27
28
      (define get-alphabet-index (lambda (val) (index-of alphabet (list-ref sequence val))))
29
      ; Iterating over the linguistic sequence,
       checking the letter precedence and counting the pairs into the matrix
      (for ((i (in-range (sub1 (length sequence)))))
33
        (let* ([row (get-alphabet-index i)]
               [col (get-alphabet-index (add1 i))]
34
               [counter (vector-ref (vector-ref matrix row) col)])
35
          (vector-set! (vector-ref matrix row) col (add1 counter))))
37
38
      matrix)
39
    ; Matrix printer function
41
42
    (define (print-matrix matrix alphabet)
      (display " ")
43
      (for-each (lambda (ch) (display (~a ch #:min-width 8 #:align 'right))) alphabet)
44
45
      (newline)
      (for ((i (in-range (vector-length matrix))))
46
47
        (printf "~a" (list-ref alphabet i))
48
        (for ((val (in-vector (vector-ref matrix i))))
49
          (display (~a val #:min-width 8 #:align 'right)))
50
        (newline)))
51
52
53
    ; Alphabet size input loop function
54
    (define (alphabet-size-input-loop)
55
      (let loop ()
56
        (display "Enter alphabet size: ")
57
        (flush-output)
        (define input (read))
59
60
        ;; Checking the alphabet size
61
        (if (> input 26)
            (begin (display "Number is too large, try again\n") (loop))
62
63
            input)))
64
65
    ; Main program
66
    (define (main)
67
      ; Reading alphabet size and constructing the alphabet
```

```
(define alphabet-size (alphabet-size-input-loop))
       (define alphabet (build-alphabet alphabet-size))
69
70
71
       ; Reading number sequence from the file and copy-sorting it
       (define filename "data.txt")
72
       (define numbers (file->list filename))
73
74
       (define sorted (sort numbers <))</pre>
75
76
       ; Building a linguistic sequence and a precedence matrix
77
       (define linguistic-sequence (build-linguistic-sequence numbers sorted alphabet))
78
       (define precedence-matrix (build-precedence-matrix linguistic-sequence alphabet))
79
80
       ; Outputting the data
       (printf "Alphabet Size: ~a\n" alphabet-size)
81
       (printf "Alphabet Size. ~a\n" (string-join alphabet ""))
(printf "Linguistic Sequence: ~a\n" (string-join linguistic-sequence ""))
(printf "Precefence Matrix:\n")
82
83
84
85
       (print-matrix precedence-matrix alphabet))
86
87
    (main)
```