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

Лабораторна робота № 1 з дисципліни «Мультипарадигмене програмування»

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

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.

Мова програмування

Fortran.

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. Читання числового ряду з файлу
 - Визначення кількості чисел у файлі (n).
 - Динамічна алокація мап'яті масиву numbers(n)
 - Читання чисел у масив.
- 2. Сортування чисел у порядку зростання
- 3. Розбиття чисел на інтервали на рівномірні інтервали
 - Визначення мінімального (min number) та максимального (max number) чисел.
 - Обчислення довжини інтервалу: $interval_length = \frac{max_number max_number}{alphabet_size}$
- 4. Генерування лінгвістичного ряду
 - Перетворення кожного числа на літеру за індексом інтервалу.
- 5. Побудова матриці передування
 - Створення квадратної матриці transition_matrix(alphabet_size, alphabet_size), де кожна комірка містить кількість випадків слідування літери стовпчика за літерою рядка.
- 6. Виведення результатів
 - Виведення лінгвістичного ряду.
 - Виведення матриці передування.

Результати виконання

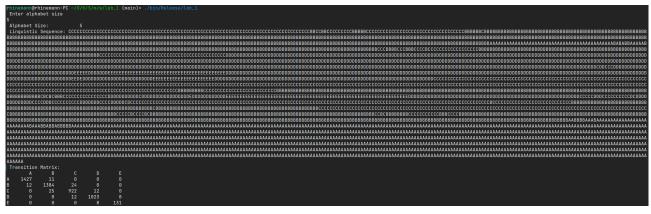


Рисунок 1: Перший числовий ряд B-C-D-E-F-Brent Oil Futures Historical Data (5000 значень - 5 символів)

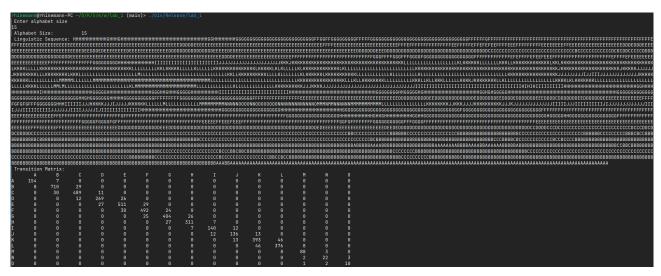


Рисунок 2: Перший числовий ряд B-C-D-E-F-Brent Oil Futures Historical Data (5000 значень - 15 символів)

Лістинг коду

```
module procedures
        implicit none
        private
5
        public quicksort, print_matrix
    contains
7
        ! Array quicksort subroutine
        recursive subroutine quicksort(a)
10
            real, intent(inout) :: a(:)
11
            real x, t
            integer :: first = 1, last
13
            integer i, j
14
15
            last = size(a, 1)
16
            x = a((first+last) / 2)
            i = first
18
            j = last
10
20
            do
21
                do while (a(i) < x)
22
                    i=i+1
23
                end do
24
                do while (x < a(j))
25
                    j=j-1
                end do
26
27
                if (i \ge j) exit
28
                t = a(i); a(i) = a(j); a(j) = t
29
                i=i+1
                j=j-1
            end do
31
32
33
            if (first < i - 1) call quicksort(a(first : i - 1))</pre>
            if (j + 1 < last) call quicksort(a(j + 1 : last))</pre>
34
35
        end subroutine quicksort
36
37
        ! Matrix Printing subroutine
38
        subroutine print matrix(matrix, alphabet)
39
            integer, intent(in) :: matrix(:,:)
40
            character(len=1), intent(in) :: alphabet(:)
41
            integer :: i
42
43
44
            print '(X, *(A8))', alphabet
45
            do i = 1, size(matrix, 1)
46
                print '(A, *(I8))', alphabet(i), matrix(i,:)
47
            end do
48
        end subroutine print_matrix
49 end module procedures
50
51
   program lab_1
52
        use procedures
53
        implicit none
54
55
        real, allocatable :: numbers(:), original_numbers(:)
56
        character(len=1), allocatable :: alphabet(:), linguistic_sequence(:)
57
        integer, allocatable :: precedence_matrix(:,:)
58
59
        integer :: n, i, index, alphabet_size, io_status, row, col
60
        real :: min_number, max_number, interval_length
61
        character(len=8) :: filename = "data.txt"
62
63
        ! Input alphabet size
64
        print *, "Enter alphabet size"
65
            read *, alphabet_size
66
67
            if (alphabet_size <= 26) exit</pre>
```

```
print *, "Alphabet size too large, try again"
69
         end do
 70
         ! Allocate alphabet and precedence matrix
 72
         allocate(alphabet(alphabet size))
 74
         allocate(precedence_matrix(alphabet_size, alphabet_size))
 75
 76
         ! Fill the alphabet with values
 77
         do i = 1, alphabet_size
             alphabet(i) = achar(64 + i)
 79
         end do
80
81
         ! Reading a number file and counting the amount of numbers
82
         open(unit=10, file=filename, status="old", action="read", iostat=io_status)
83
         if (io_status /= 0) then
             print *, "Error opening file."
84
85
             stop
         end if
86
87
88
         n = 0
89
         do
90
             read(10, *, iostat=io_status)
             if (io_status /= 0) exit
91
92
             n = n + 1
93
         end do
94
         close(10)
95
96
         ! Allocating memory for arrays
97
         allocate(numbers(n))
98
         allocate(original_numbers(n))
99
         allocate(linguistic_sequence(n))
100
         ! Reading numbers from file to original_numbers
         open(unit=10, file=filename, status="old", action="read", iostat=io_status)
         if (io_status /= 0) then
             print *, "Error opening file."
104
             stop
106
         end if
107
         do i = 1, n
109
             read(10, *) original_numbers(i)
110
         end do
         close(10)
111
113
         ! Copying original numbers to numbers for sorting
         numbers = original numbers
114
115
         call quicksort(numbers)
116
117
         ! Calculating step intervals
118
         min_number = numbers(1)
         max number = numbers(n)
119
         interval_length = (max_number - min_number) / alphabet_size
121
         ! Converting numbers to a lexical string
123
         do i = 1, n
             index = ceiling((original_numbers(i) - min_number) / interval_length)
124
125
             if (index < 1) index = 1
126
             if (index > alphabet_size) index = alphabet_size
127
             linguistic_sequence(i) = alphabet(index)
128
         end do
129
130
         ! Constructing a precedence matrix
131
         do i = 1, n - 1
             row = findloc(alphabet, linguistic_sequence(i), 1)
             col = findloc(alphabet, linguistic_sequence(i + 1), 1)
133
134
             precedence matrix(row, col) = precedence matrix(row, col) + 1
135
         end do
136
137
         ! Printing program results
         print *, "Alphabet Size:", alphabet_size
138
```

```
print *, "Linguistic Sequence:", " ", linguistic_sequence
print *, "Transition Matrix:"
call print_matrix(precedence_matrix, alphabet)

! Freeing allocated memory
deallocate(numbers, original_numbers, alphabet, linguistic_sequence, precedence_matrix)
end program lab_1
```