Отчёт по лабораторной работе №2
Структуры данных

Статический анализ данных

Выполнила: Коняева Марина Александровна,
НФИбд-01-21, 1032217044

Содержание

Список иллюстраций

Цели лабораторной работы

Изучить несколько структур данных, реализованных в Julia, научиться применять их и операции над ними для решения задач.

Теоретическое введение

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения.[1]

Рассмотрим несколько структур данных, реализованных в Julia. Несколько функций (методов), общих для всех структур данных: – isempty() — проверяет, пуста ли структура данных; – length() — возвращает длину структуры данных; – in() — проверяет принадлежность элемента к структуре; – unique() — возвращает коллекцию уникальных элементов структуры, – reduce() — свёртывает структуру данных в соответствии с заданным бинарным оператором; – maximum() (или minimum()) — возвращает наибольший (или наименьший) результат вызова функции для каждого элемента структуры данных.

Задачи лабораторной работы

1. Используя Jupyter Lab, повторите примеры из раздела 2.2.
2. Выполните задания для самостоятельной работы (раздел 2.4).

Выполнение лабораторной работы

Кортежи

1. Изучим информацию о кортежах (Tuple), структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы). Синтаксис определения кортежа: (element1, element2, …).

2. Повторим примеры с кортежами, а именно унаем как их определять и какие типы элементов он может содержать (пустой кортеж, котреж из элементов целых чисел/String/разных типов).

```
# пустой кортеж:
()
```

```
()
```

```
# кортеж из элементов типа String:
favoritelang = ("Python","Julia","R")
```

```
("Python", "Julia", "R")
```

```
# кортеж из целых чисел:
x1 = (1, 2, 3)
```

```
(1, 2, 3)
```

```
# кортеж из элементов разных типов:
x2 = (1, 2.0, "tmp")
```

```
(1, 2.0, "tmp")
```

```
# именованный кортеж:
x3 = (a=2, b=1+2)
```

```
(a = 2, b = 3)
```

Рис. 1: Примеры кортежей:

3. Повторим примеры операций над кортежами (находение длины, обращение к элементам кортежа, сложение элементов, проверка вхождения).

```
# длина кортежа x2:
length(x2)
```

```
3
```

```
# обратиться к элементам кортежа x2:
x2[1], x2[2], x2[3]
```

```
(1, 2.0, "tmp")
```

```
# произвести какую-либо операцию (сложение)
# с вторым и третьим элементами кортежа x1:
c = x1[2] + x1[3]
```

```
5
```

```
# обращение к элементам именованного кортежа x3:
x3.a, x3.b, x3[2]
```

```
(2, 3, 3)
```

```
# проверка вхождения элементов tmp и 0 в кортеж x2
# (два способа обращения к методу in()):
in("tmp", x2), 0 in x2
```

```
(true, false)
```

Рис. 2: Примеры операций над кортежами:

Словари

4. Изучим информацию о словарях, неупорядоченный набор связанных между собой по ключу данных. Синтаксис определения словаря: Dict(key1 => value1, key2 => value2, …).

5. Повторим примеры со словарями, а именно изучим, как их создавать, вывести ключи, значения, пары (ключ - значение), проверку вхождения.

```
# создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309","333-5544"),"Бухгалтерия" => "555-2368")
```

```
Dict{String, Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")
```

```
# вывести ключи словаря:
keys(phonebook)
```

```
KeySet for a Dict{String, Any} with 2 entries. Keys:
  "Бухгалтерия"
  "Иванов И.И."
```

```
# вывести значения элементов словаря:
values(phonebook)
```

```
ValueIterator for a Dict{String, Any} with 2 entries. Values:
  "555-2368"
  ("867-5309", "333-5544")
```

```
# вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)
```

```
Dict{String, Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")
```

```
# проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")
```

```
true
```

Рис. 3: Примеры словарей и операций над ними 1

6. Повторим примеры со словарями, а именно изучим добавление и удаление элементов, обьединение словарей.

```
# добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"
```

```
"555-3344"
```

```
# удалить ключ и связанные с ним значения из словаря
pop!(phonebook, "Иванов И.И.")
```

```
("867-5309", "333-5544")
```

```
# Объединение словарей (функция merge()):
a = Dict("foo" => 0.0, "bar" => 42.0);
b = Dict("baz" => 17, "bar" => 13.0);
merge(a, b), merge(b,a)
```

```
(Dict{String, Real}("bar" => 13.0, "baz" => 17, "foo" => 0.0), Dict{String, Real}("bar" => 42.0, "baz" => 17, "foo" => 0.0))
```

Рис. 4: Примеры словарей и операций над ними 2

Множества

7. Изучим информацию о множествах, как структуре данных в Julia, соответствует множеству, как математическому объекту, то есть является неупорядоченной совокупностью элементов какого-либо типа. Возможные операции над множествами: объединение, пересечение, разность; принадлежность элемента множеству. Синтаксис определения множества: Set([itr]), где itr — набор значений, сгенерированных данным итерируемым объектом или пустое множество.

8. Повторим примеры множеств и операций над ними, а именно как задать множество и его значения, объединение, проверка эквивалентности, разноть, проверка вхождения, добавление и удаление элемента.

```
# создать множество из четырёх целочисленных значений:
A = Set([1, 3, 4, 5])
```

```
Set{Int64} with 4 elements:
  5
  4
  3
  1
```

```
# создать множество из 11 символьных значений:
B = Set("abrakadabra")
```

```
Set{Char} with 5 elements:
  'a'
  'd'
  'r'
  'k'
  'b'
```

```
# проверка эквивалентности двух множеств:
S1 = Set([1,2]);
S2 = Set([3,4]);
issetequal(S1,S2)
```

```
false
```

```
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
issetequal(S3,S4)
```

```
true
```

Рис. 5: Примеры множеств и операций над ними 1

```
# объединение множеств:
C=union(S1,S2)
```

```
Set{Int64} with 4 elements:
  4
  2
  3
  1
```

```
# пересечение множеств:
D = intersect(S1,S3)
```

```
Set{Int64} with 2 elements:
  2
  1
```

```
# разность множеств:
E = setdiff(S3,S1)
```

```
Set{Int64} with 1 element:
  3
```

```
# проверка вхождения элементов одного множества в другое:
issubset(S1,S4)
```

```
true
```

Рис. 6: Примеры множеств и операций над ними 2

```
# добавление элемента в множество:
push!(S4, 99)
```

```
Set{Int64} with 4 elements:
  2
  99
  3
  1
```

```
# удаление последнего элемента множества:
pop!(S4)
```

```
2
```

Рис. 7: Примеры множеств и операций над ними 3

Массивы

9. Изучим информацию о массивах, коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов. Общий синтаксис одномерных массивов: array_name_1 = [element1, element2, …] array_name_2 = [element1 element2 …].

10. Повторим примеры массивов: создание пуского массива с абстрактным/конкретным типом, вектор-столбец, вектор-строка, многомерные массивы.

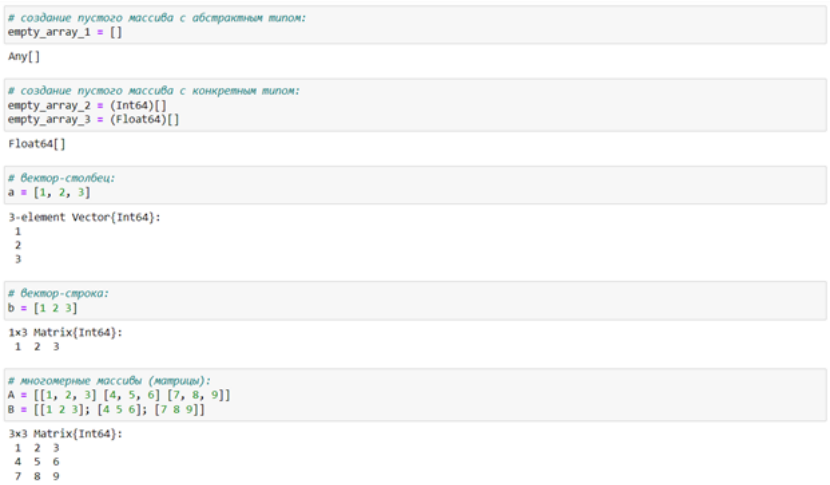

Рис. 8: Примеры массивов 1

одномерный массив из 8 элементов (массив 1×8) со значениями, случайно распределёнными на интервале [0, 1):

```
c = rand(1,8)
```

```
1×8 Matrix{Float64}:
 0.390644  0.00995584  0.793286  0.700352  …  0.184297  0.791407  0.598328
```

многомерный массив 2×3 (2 строки, 3 столбца) элементов со значениями, случайно распределёнными на интервале [0, 1):

```
C = rand(2,3)
```

```
2×3 Matrix{Float64}:
 0.916608  0.993444  0.500424
 0.800927  0.118592  0.119084
```

```
# трёхмерный массив:
D = rand(4, 3, 2)
```

```
4×3×2 Array{Float64, 3}:
[:, :, 1] =
 0.778071  0.864328  0.259164
 0.515813  0.662949  0.647521
 0.633793  0.956924  0.94884
 0.189616  0.949397  0.95729

[:, :, 2] =
 0.980165  0.845072  0.455935
 0.896212  0.235379  0.631316
 0.233784  0.137664  0.515154
 0.968408  0.130949  0.548336
```

Рис. 9: Примеры массивов 2

11. Повторим примеры массивов, заданных некоторыми функциями через включение.

```
# массив из квадратных корней всех целых чисел от 1 до 10:
roots = [sqrt(i) for i in 1:10]
```

```
10-element Vector{Float64}:
 1.0
 1.4142135623730951
 1.7320508075688772
 2.0
 2.23606797749979
 2.449489742783178
 2.6457513110645907
 2.8284271247461903
 3.0
 3.1622776601683795
```

```
# массив с элементами вида 3*x^2,
# где x - нечётное число от 1 до 9 (включительно)
ar_1 = [3*i^2 for i in 1:2:9]
```

```
5-element Vector{Int64}:
   3
  27
  75
 147
 243
```

```
# массив квадратов элементов, если квадрат не делится на 5 или 4:
ar_2 = [i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]
```

```
4-element Vector{Int64}:
  1
  9
 49
 81
```

Рис. 10: Примеры массивов, заданных некоторыми функциями через включени

12. Изучим некоторые операции для работы с массивами: – length(A) — число элементов массива A; – ndims(A) — число размерностей массива A; – size(A) — кортеж размерностей массива A; – size(A, n) — размерность массива A в заданном направлении; – copy(A) — создание копии массива A; – ones(), zeros() — создать массив с единицами или нулями соответственно; – fill(value,array_name) — заполнение массива заранее определенным значением; – sort() — сортировка элементов;

– collect() — вернуть массив всех элементов в коллекции или итераторе; – reshape() — изменение размера массива; – transpose() — транспонирование массива.

13. Потворим несколько примеров операций для работы с массивами.

```
# одномерный массив из пяти единиц:
ones(5)
```

```
5-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0
```

```
# двумерный массив 2x3 из единиц:
ones(2,3)
```

```
2×3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0
```

```
# одномерный массив из 4 нулей:
zeros(4)
```

```
4-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
```

```
# заполнить массив 3x2 цифрами 3.5
fill(3.5,(3,2))
```

```
3×2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5
```

Рис. 11: Некоторые операции для работы с массивами 1

```
# заполнение массива посредством функции repeat():
repeat([1,2],3,3)
repeat([1 2],3,3)
```

```
3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

```
# преобразование одномерного массива из целых чисел от 1 до 12
# в двумерный массив 2x6
a = collect(1:12)
print(a)
print()
b = reshape(a,(2,6))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
2×6 Matrix{Int64}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

```
# транспонирование b
c = transpose(b)
```

```
6×2 transpose(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

Рис. 12: Некоторые операции для работы с массивами 1

```
# массив 10x5 целых чисел в диапазоне [10, 20]:
ar = rand(10:20, 10, 5)
```

```
10×5 Matrix{Int64}:
 18  15  20  20  11
 14  14  20  16  15
 12  15  19  16  14
 19  17  20  12  11
 11  13  12  11  17
 17  12  17  14  13
 16  11  15  12  16
 18  12  10  17  13
 10  19  16  18  20
 14  18  20  17  20
```

```
# выбор всех значений строки в столбце 2:
ar[:, 2]
```

```
10-element Vector{Int64}:
 15
 14
 15
 17
 13
 12
 11
 12
 19
 18
```

Рис. 13: Некоторые операции для работы с массивами 1

```
# выбор всех значений в столбцах 2 и 5:
ar[:, [2, 5]]
```

```
10×2 Matrix{Int64}:
 15  11
 14  15
 15  14
 17  11
 13  17
 12  13
 11  16
 12  13
 19  20
 18  20
```

```
# все значения строк в столбцах 2, 3 и 4:
ar[:, 2:4]
```

```
10×3 Matrix{Int64}:
 15  20  20
 14  20  16
 15  19  16
 17  20  12
 13  12  11
 12  17  14
 11  15  12
 12  10  17
 19  16  18
 18  20  17
```

Рис. 14: Некоторые операции для работы с массивами 1

```
# значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]
```

```
3×2 Matrix{Int64}:
 14  15
 19  11
 17  13
```

```
# значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]
```

```
3-element Vector{Int64}:
 20
 20
 11
```

```
# сортировка по столбцам:
sort(ar,dims=1)
```

```
10×5 Matrix{Int64}:
 10  11  10  11  11
 11  12  12  12  11
 12  12  15  12  13
 14  13  16  14  13
 14  14  17  16  14
 16  15  19  16  15
 17  15  20  17  16
 18  17  20  17  17
 18  18  20  18  20
 19  19  20  20  20
```

Рис. 15: Некоторые операции для работы с массивами 1

```
# сортировка по строкам:
sort(ar,dims=2)
```

```
10×5 Matrix{Int64}:
 11  15  18  20  20
 14  14  15  16  20
 12  14  15  16  19
 11  12  17  19  20
 11  11  12  13  17
 12  13  14  17  17
 11  12  15  16  16
 10  12  13  17  18
 10  16  18  19  20
 14  17  18  20  20
```

```
# поэлементное сравнение с числом
# (результат - массив логических значений):
ar .> 14
```

```
10×5 BitMatrix:
 1  1  1  1  0
 0  0  1  1  1
 0  1  1  1  0
 1  1  1  0  0
 0  0  0  0  1
 1  0  1  0  0
 1  0  1  0  1
 1  0  0  1  0
 0  1  1  1  1
 0  1  1  1  1
```

Рис. 16: Некоторые операции для работы с массивами 1

```
# возврат индексов элементов массива, удовлетворяющих условию:
findall(ar .> 14)
```

```
29-element Vector{CartesianIndex{2}}:
 CartesianIndex(1, 1)
 CartesianIndex(4, 1)
 CartesianIndex(6, 1)
 CartesianIndex(7, 1)
 CartesianIndex(8, 1)
 CartesianIndex(1, 2)
 CartesianIndex(3, 2)
 CartesianIndex(4, 2)
 CartesianIndex(9, 2)
 CartesianIndex(10, 2)
 CartesianIndex(1, 3)
 CartesianIndex(2, 3)
 CartesianIndex(3, 3)
 ⋮
 CartesianIndex(10, 3)
 CartesianIndex(1, 4)
 CartesianIndex(2, 4)
 CartesianIndex(3, 4)
 CartesianIndex(8, 4)
 CartesianIndex(9, 4)
 CartesianIndex(10, 4)
 CartesianIndex(2, 5)
 CartesianIndex(5, 5)
 CartesianIndex(7, 5)
 CartesianIndex(9, 5)
 CartesianIndex(10, 5)
```

Рис. 17: Некоторые операции для работы с массивами 1

Самостоятельная работа

14. Выполним 1 задание для самостойтельной работы: даны множества $A = { 0, 3, 4, 9 } $, $B = \{1, 3, 4, 7\}$ и $C = \{0, 1, 2, 4, 7, 8, 9\}$. Найти $P = (A \cap B) \cup (A \cap B) \cup (A \cap C) \cup (B \cap C)$.

```
function peresech(x,y)
    z = []
    for i in x
        for j in y
            if i == j
                append!(z,i)
            end
        end
    end
    return z
end


function obyedin(x,y...)
    z = x
    for i in y
        for j in i
            if findfirst(isequal(j),z) == nothing
                append!(z, j)
            end
        end
    end
    return sort(z)
end

A = [0, 3, 4, 9]
B = [1, 3, 4, 7]
C = [0, 1, 2, 4, 7, 8, 9]

peresech(A, B), obyedin(A, B)
```

```
(Any[3, 4], [0, 1, 3, 4, 7, 9])
```

Рис. 18: Выполнение 1 задания: вариант 1

```
t1 = peresech(A, B)
t2 = peresech(A, C)
t3 = peresech(B, C)
P = obyedin(t1, t1, t2, t3)
```

```
6-element Vector{Any}:
 0
 1
 3
 4
 7
 9
```

Рис. 19: Выполнение 1 задания: вариант 1

```
a = Set([0, 3, 4, 9]); b = Set(B); c = Set(C)
a, b, c
```

```
(Set([0, 4, 9, 3]), Set([4, 7, 3, 1]), Set([0, 4, 7, 2, 9, 8, 1]))
```

```
intersect(a,b), union(a,b)
```

```
(Set([4, 3]), Set([0, 4, 7, 9, 3, 1]))
```

```
t1 = intersect(a,b)
t2 = intersect(a,c)
t3 = intersect(b,c)
P = union(t1, t1, t2, t3)
```

```
Set{Int64} with 6 elements:
  0
  4
  7
  9
  3
  1
```

Рис. 20: Выполнение 1 задания: вариант 2

15. Приведите свои примеры с выполнением операций над множествами элементов разных типов.

```
t = Set([1 2 3; 4 5 6; "Ma" "ri" "Marina"])
```

```
Set{Any} with 9 elements:
  5
  4
  6
  "Marina"
  2
  "Ma"
  "ri"
  3
  1
```

```
1 in t
```

```
true
```

```
0 in t
```

```
false
```

```
[1, 2, 3] in t
```

```
false
```

```
intersect(Set([1,2,3,4,7,8,10]), t)
```

```
Set{Any} with 4 elements:
  4
  2
  3
  1
```

Рис. 21: Выполнение 2 задания

```
union(Set([1,2,3,4,7,8,10]), t)
```

```
Set{Any} with 12 elements:
  5
  7
  "Marina"
  "Ma"
  8
  1
  4
  6
  2
  10
  "ri"
  3
```

```
setdiff(Set([1,2,3,4,7,8,10]), t)
```

```
Set{Int64} with 3 elements:
  7
  10
  8
```

```
setdiff(t, Set([1,2,3,4,7,8,10]))
```

```
Set{Any} with 5 elements:
  5
  6
  "Marina"
  "Ma"
  "ri"
```

Рис. 22: Выполнение 2 задания

16. Выполним 3 задание: создать разными способами несколько видов массивов:

1) массив (1, 2, 3, … □ − 1, □), □ выберите больше 20;

```
N = 100
t = zeros(Int64, N)
for i in 1:N
    t[i] = i
end
t
```

```
100-element Vector{Int64}:
   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12
  13
   ⋮
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
 100
```

Рис. 23: Массив: вариант 1

```
t = collect(1:N)
t
```

```
100-element Vector{Int64}:
   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12
  13
   ⋮
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
 100
```

Рис. 24: Массив: вариант 2

```
t = [i for i in 1:N]
```

```
100-element Vector{Int64}:
   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12
  13
   ⋮
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
 100
```

Рис. 25: Массив: вариант 3

2) массив (□, □ − 1 … , 2, 1), □ выберите больше 20;

```
N = 100
t = zeros(Int64, N)
for i in N:-1:1
    t[N-i+1] = i
end
t
```

```
100-element Vector{Int64}:
 100
  99
  98
  97
  96
  95
  94
  93
  92
  91
  90
  89
  88
   ⋮
  12
  11
  10
   9
   8
   7
   6
   5
   4
   3
   2
   1
```

Рис. 26: Массив: вариант 1

```
t = collect(N:-1:1)
```

```
100-element Vector{Int64}:
 100
  99
  98
  97
  96
  95
  94
  93
  92
  91
  90
  89
  88
   ⋮
  12
  11
  10
   9
   8
   7
   6
   5
   4
   3
   2
   1
```

Рис. 27: Массив: вариант 2

```
t = [N-i+1 for i in 1:N]
```

```
100-element Vector{Int64}:
 100
  99
  98
  97
  96
  95
  94
  93
  92
  91
  90
  89
  88
   ⋮
  12
  11
  10
   9
   8
   7
   6
   5
   4
   3
   2
   1
```

Рис. 28: Массив: вариант 3

3) массив (1, 2, 3, … , □ − 1, □, □ − 1, … , 2, 1), □ выберите больше 20;

```
t = zeros(Int64, 2N-1)
for i in 1:N
    t[i] = i
end
for i in 1:N-1
    t[i+N] = N-i
end
t
```

```
199-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

Рис. 29: Массив: вариант 1

```
t = cat([i for i in 1:N], [i for i in N-1:-1:1], dims=1)
```

```
199-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

Рис. 30: Массив: вариант 2

```
t = vcat(collect(1:N), collect(N:-1:1))
```

```
200-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

Рис. 31: Массив: вариант 3

4) массив с именем tmp вида (4, 6, 3);

```
t1 = 4; t2 = 6; t3 = 3
tmp = [t1, t2, t3]
```

```
3-element Vector{Int64}:
 4
 6
 3
```

```
tmp = [4, 6, 3]
```

```
3-element Vector{Int64}:
 4
 6
 3
```

Рис. 32: Массив с именем tmp (несколько вариантов)

5) массив, в котором первый элемент массива tmp повторяется 10 раз;

```
t1 = zeros(Int64, 10)
for i in 1:10
    t1[i] = tmp[1]
end
t1
```

```
10-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
```

```
t1 = [tmp[1] for i in 1:10]
```

```
10-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
```

```
t1 = fill(tmp[1], 10)
```

```
10-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
```

Рис. 33: Массив (несколько вариантов)

6) массив, в котором все элементы массива tmp повторяются 10 раз;

```
t2 = zeros(Int64, 30)
for i in 0:9
    t2[3*i+1] = tmp[1]
    t2[3*i+2] = tmp[2]
    t2[3*i+3] = tmp[3]
end
t2
```

```
30-element Vector{Int64}:
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 ⋮
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
```

Рис. 34: Массив: вариант 1

```
t2 = repeat(tmp, 10)
```

```
30-element Vector{Int64}:
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 ⋮
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
```

Рис. 35: Массив: вариант 2

7) массив, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз;

```
t3 = [tmp[1] for i in 1:11]
for i in 1:10
    append!(t3, tmp[2])
    append!(t3, tmp[3])
end
t3
```

```
31-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 3
 ⋮
 6
 3
 6
 3
 6
 3
 6
 3
 6
 3
 6
 3
```

Рис. 36: Массив: вариант 1

```
t3 = repeat(tmp, 10)
append!(t3, tmp[1])
t3
```

```
31-element Vector{Int64}:
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 ⋮
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
```

Рис. 37: Массив: вариант 2

```
t3 = cat([tmp[1] for i in 1:11], [tmp[2] for i in 1:10], [tmp[3] for i in 1:10], dims=1)
```

```
31-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 6
 ⋮
 6
 6
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

Рис. 38: Массив: вариант 3

8) массив, в котором первый элемент массива tmp встречается 10 раз подряд, второй элемент — 20 раз подряд, третий элемент — 30 раз подряд;

```
t3 = [tmp[1] for i in 1:10]
for i in 1:20
    append!(t3, tmp[2])
    append!(t3, tmp[3])
end
for i in 1:10
    append!(t3, tmp[3])
end
t3
```

```
60-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 3
 6
 ⋮
 6
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

Рис. 39: Массив: вариант 1

```
t3 = repeat(tmp, 10)
for i in 1:10
    append!(t3, tmp[2])
    append!(t3, tmp[3])
end
for i in 1:10
    append!(t3, tmp[3])
end
t3
```

```
60-element Vector{Int64}:
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 ⋮
 6
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

Рис. 40: Массив: вариант 2

```
t3 = cat([tmp[1] for i in 1:10], [tmp[2] for i in 1:20], [tmp[3] for i in 1:30], dims=1)
```

```
60-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 6
 6
 ⋮
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

Рис. 41: Массив: вариант 3

9) массив из элементов вида $2^{tmp[i]}$, $i = 1, 2, 3$, где элемент $2^{tmp[i]}$ встречается 4 раза; посчитайте в полученном векторе, сколько раз встречается цифра 6, и выведите это значение на экран;

```
t2 = []
tmpsize = size(tmp)[1]
for i in 1:4
    for j in 1:tmpsize
        append!(t2, 2^(tmp[j]))
    end
end
t2
```

```
12-element Vector{Any}:
 16
 64
  8
 16
 64
  8
 16
 64
  8
 16
 64
  8
```

```
t2 = fill(2, 4*tmpsize)
for i in 0:3
    for j in 1:3
        t2[i*tmpsize+j] = t2[i*tmpsize+j]^tmp[j]
    end
end
t2
```

```
12-element Vector{Int64}:
 16
 64
  8
 16
 64
  8
 16
 64
  8
 16
 64
  8
```

Рис. 42: Массив (1-2 вариант)

```
t2 = repeat(fill(2, tmpsize).^tmp,4)
```

```
12-element Vector{Int64}:
 16
 64
  8
 16
 64
  8
 16
 64
  8
 16
 64
  8
```

```
size(findall(isequal("6"), split(join(string.(t2)), "")))[1]
```

```
8
```

```
count("6", join(string.(t2)))
```

```
8
```

Рис. 43: Массив (вариант 3) и подсчеты цифры 6

10) вектор значений $y = e^x \cdot cos(x)$ в точках $x = 3, 3.1, 3.2, ..., 6$, найдите среднее значение y;

```
t1 = collect(3:0.1:6)
t = ones(size(t1)[1])
for i in 1:size(t1)[1]
    t[i]= exp(t1[i])*cos(t1[i])
end
t
```

```
31-element Vector{Float64}:
 -19.884530844146987
 -22.178753389342127
 -24.490696732801293
 -26.77318244299338
 -28.969237768093574
 -31.011186439374516
 -32.819774760338504
 -34.30336011037369
 -35.35719361853035
 -35.86283371230767
 -35.68773248011913
 -34.68504225166807
 -32.693695428321746
   ⋮
  25.046704998273004
  42.09920106253839
  61.99663027669454
  84.92906736250268
 111.0615860420258
 140.5250750527875
 173.40577640857734
 209.73349424783467
 249.46844055885668
 292.4867067371223
 338.5643778585117
 387.36034029093076
```

Рис. 44: Вектор: вариант 1

```
t = [exp(i)*cos(i) for i in 3:0.1:6]
```

```
31-element Vector{Float64}:
 -19.884530844146987
 -22.178753389342127
 -24.490696732801293
 -26.77318244299338
 -28.969237768093574
 -31.011186439374516
 -32.819774760338504
 -34.30336011037369
 -35.35719361853035
 -35.86283371230767
 -35.68773248011913
 -34.68504225166807
 -32.693695428321746
   ⋮
  25.046704998273004
  42.09920106253839
  61.99663027669454
  84.92906736250268
 111.0615860420258
 140.5250750527875
 173.40577640857734
 209.73349424783467
 249.46844055885668
 292.4867067371223
 338.5643778585117
 387.36034029093076
```

```
sum(t)/size(t)[1]
```

```
53.11374594642971
```

Рис. 45: Вектор (вариант 2) и сумма

11) вектор вида $(x^i, y^j), x = 0.1, \quad i = 3, 6, 9, ..., 36, \quad y = 0.2, \quad j = 1, 4, 7, ..., 34;$

```
t1 = collect(3:3:36)
t2 = collect(1:3:34)
t = zeros(size(t1)[1],2)
for i in 1:size(t1)[1]
    t[i,1] = 0.1^t1[i]
    t[i,2] = 0.2^t2[i]
end
t
```

```
12×2 Matrix{Float64}:
 0.001    0.2
 1.0e-6   0.0016
 1.0e-9   1.28e-5
 1.0e-12  1.024e-7
 1.0e-15  8.192e-10
 1.0e-18  6.5536e-12
 1.0e-21  5.24288e-14
 1.0e-24  4.1943e-16
 1.0e-27  3.35544e-18
 1.0e-30  2.68435e-20
 1.0e-33  2.14748e-22
 1.0e-36  1.71799e-24
```

```
t = zeros(size(collect(3:3:36))[1],2)
for i in 0:size(collect(3:3:36))[1]-1
    t[i+1,1] = 0.1^(3*(i+1))
    t[i+1,2] = 0.2^(3*i+1)
end
t
```

```
12×2 Matrix{Float64}:
 0.001    0.2
 1.0e-6   0.0016
 1.0e-9   1.28e-5
 1.0e-12  1.024e-7
 1.0e-15  8.192e-10
 1.0e-18  6.5536e-12
 1.0e-21  5.24288e-14
 1.0e-24  4.1943e-16
 1.0e-27  3.35544e-18
 1.0e-30  2.68435e-20
 1.0e-33  2.14748e-22
 1.0e-36  1.71799e-24
```

Рис. 46: Вектор (1-2 вариант)

```
t = hcat(fill(0.1, size(collect(3:3:36))[1]), fill(0.2, size(collect(1:3:34))[1]))
t[:, 1] = t[:, 1].^collect(3:3:36)
t[:, 2] = t[:, 2].^collect(1:3:34)
t
```

```
12×2 Matrix{Float64}:
 0.001    0.2
 1.0e-6   0.0016
 1.0e-9   1.28e-5
 1.0e-12  1.024e-7
 1.0e-15  8.192e-10
 1.0e-18  6.5536e-12
 1.0e-21  5.24288e-14
 1.0e-24  4.1943e-16
 1.0e-27  3.35544e-18
 1.0e-30  2.68435e-20
 1.0e-33  2.14748e-22
 1.0e-36  1.71799e-24
```

Рис. 47: Вектор: вариант 3

12) вектор с элементами $\frac{2^i}{i}$, $i = 1, 2, ..., M$, $M = 25$;

```
t3 = cat([tmp[1] for i in 1:11], [tmp[2] for i in 1:10], [tmp[3] for i in 1:10], dims=1)
```

```
31-element Vector{Int64}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 6
 ⋮
 6
 6
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

Рис. 48: Вектор: вариант 1

```
t = zeros(M)
for i in 1:M
    t[i] = 2^i/i
end
t
```

```
25-element Vector{Float64}:
      2.0
      2.0
      2.6666666666666665
      4.0
      6.4
     10.666666666666666
     18.285714285714285
     32.0
     56.888888888888886
    102.4
    186.1818181818182
    341.3333333333333
    630.1538461538462
   1170.2857142857142
   2184.5333333333333
   4096.0
   7710.117647058823
  14563.555555555555
  27594.105263157893
  52428.8
  99864.38095238095
 190650.18181818182
 364722.0869565217
 699050.6666666666
      1.34217728e6
```

Рис. 49: Вектор: вариант 2

```
t = fill(2, M).^collect(1:M)./collect(1:M)
t
```

```
25-element Vector{Float64}:
      2.0
      2.0
      2.6666666666666665
      4.0
      6.4
     10.666666666666666
     18.285714285714285
     32.0
     56.888888888888886
    102.4
    186.1818181818182
    341.3333333333333
    630.1538461538462
   1170.2857142857142
   2184.5333333333333
   4096.0
   7710.117647058823
  14563.555555555555
  27594.105263157893
  52428.8
  99864.38095238095
 190650.18181818182
 364722.0869565217
 699050.6666666666
      1.34217728e6
```

Рис. 50: Вектор: вариант 3

13) вектор вида (“fn1”, “fn2”, …, “fnN”), $N = 30$;

```
N = 30
t = [join(["fn", string(i)]) for i in 1:N]
```

```
30-element Vector{String}:
 "fn1"
 "fn2"
 "fn3"
 "fn4"
 "fn5"
 "fn6"
 "fn7"
 "fn8"
 "fn9"
 "fn10"
 "fn11"
 "fn12"
 "fn13"
 ⋮
 "fn19"
 "fn20"
 "fn21"
 "fn22"
 "fn23"
 "fn24"
 "fn25"
 "fn26"
 "fn27"
 "fn28"
 "fn29"
 "fn30"
```

Рис. 51: Вектор: вариант 1

```
t1 = string.(collect(1:N))
t2 = fill("fn", N)
t = join.([t2[i], t1[i]] for i in 1:N)
```

```
30-element Vector{String}:
 "fn1"
 "fn2"
 "fn3"
 "fn4"
 "fn5"
 "fn6"
 "fn7"
 "fn8"
 "fn9"
 "fn10"
 "fn11"
 "fn12"
 "fn13"
 ⋮
 "fn19"
 "fn20"
 "fn21"
 "fn22"
 "fn23"
 "fn24"
 "fn25"
 "fn26"
 "fn27"
 "fn28"
 "fn29"
 "fn30"
```

Рис. 52: Вектор: вариант 2

14) векторы $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$ целочисленного типа длины $n = 250$ как случайные выборки из совокупности $0, 1, ..., 999$ и на их основе:

```
n = 250
x, y = rand(0:999, n), rand(0:999, n)
```

```
([503, 188, 795, 134, 497, 117, 857, 212, 469, 367  …  204, 223, 297, 573, 613, 586, 727, 846, 322, 310],
[249, 380, 695, 565, 579, 516, 460, 569, 616, 625  …  987, 561, 15, 584, 621, 246, 748, 406, 460, 74])
```

Рис. 53: Начальные условия

- сформируем вектор $(y_2 - x_1, ..., y_n - x_{n-1})$;

```
t = zeros(Int64,n-1)
for i in 1:n-1
    t[i] = y[i+1]-x[i]
end
t
```

```
249-element Vector{Int64}:
 -123
  507
 -230
  445
   19
  343
 -288
  404
  156
  622
 -183
  550
  409
    ⋮
 -486
 -339
   -7
  357
 -208
  287
   48
 -367
  162
 -321
 -386
 -248
```

Рис. 54: Вектор: вариант 1

```
t = [y[i+1]-x[i] for i in 1:n-1]
```

```
249-element Vector{Int64}:
 -123
  507
 -230
  445
   19
  343
 -288
  404
  156
  622
 -183
  550
  409
    ⋮
 -486
 -339
   -7
  357
 -208
  287
   48
 -367
  162
 -321
 -386
 -248
```

Рис. 55: Вектор: вариант 2

- сформируем вектор $(x_1 + 2x_2 - x_3, x_2 + 2x_3 - x_4, ..., x_{n-2} + 2x_{n-1} - x_n)$;

```
t = zeros(Int64, n-2)
for i in 1:n-2
    t[i] += x[i]
    t[i] += 2*x[i+1]
    t[i] -= x[i+2]
end
t
```

```
248-element Vector{Int64}:
   84
 1644
  566
 1011
 -126
 1619
  812
  783
 1016
  394
  801
  470
  -22
    ⋮
 1044
 1733
 2780
 1179
  353
  244
  830
 1213
 1058
 1194
 2097
 1180
```

Рис. 56: Вектор: вариант 1

```
t = [x[i-2]+2*x[i-1]-x[i] for i in 3:n]
```

```
248-element Vector{Int64}:
   84
 1644
  566
 1011
 -126
 1619
  812
  783
 1016
  394
  801
  470
  -22
    ⋮
 1044
 1733
 2780
 1179
  353
  244
  830
 1213
 1058
 1194
 2097
 1180
```

Рис. 57: Вектор: вариант 2

- сформируем вектор $(\frac{sin(y_1)}{cos(x_2)}, \frac{sin(y_2)}{cos(x_3)}, ..., \frac{sin(y_{n-1})}{cos(x_n)})$;

```
t = zeros(n-1)
for i in 1:n-1
    t[i] = sin(y[i])/cos(x[i+1])
end
t
```

```
249-element Vector{Float64}:
  -0.8266008568576301
  -0.13442344345843818
   1.402098237625552
  -0.5781166755611429
  -1.1208973195230876
  -0.8855938371941715
 -16.88709755854497
   0.5863265950096858
  -0.2906969009595933
   2.341696000323015
   3.891058431044566
   6.855887229247249
  -1.3027804851186098
   ⋮
  -0.035603875075865606
  -2.35376096814028
   0.40432551126998334
  -0.5147810080712839
  -8.17577910136683
   1.946336969079768
   0.35663725304408617
   9.264040747089897
  -2.968898721945723
  -0.48396061322777983
 -50.597984989853614
  -1.8474291602292727
```

Рис. 58: Вектор: вариант 1

```
t = [sin(y[i-1])/cos(x[i]) for i in 2:n]
```

```
249-element Vector{Float64}:
  -0.8266008568576301
  -0.13442344345843818
   1.402098237625552
  -0.5781166755611429
  -1.1208973195230876
  -0.8855938371941715
 -16.88709755854497
   0.5863265950096858
  -0.2906969009595933
   2.341696000323015
   3.891058431044566
   6.855887229247249
  -1.3027804851186098
   ⋮
  -0.035603875075865606
  -2.35376096814028
   0.40432551126998334
  -0.5147810080712839
  -8.17577910136683
   1.946336969079768
   0.35663725304408617
   9.264040747089897
  -2.968898721945723
  -0.48396061322777983
 -50.597984989853614
  -1.8474291602292727
```

Рис. 59: Вектор: вариант 2

- вычислите $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i+10}$;

```
summ = 0
for i in 1:n-1
    summ += exp(-x[i+1])/(x[i]+10)
end
summ
```

```
0.0004852622754867377
```

```
sum([exp(-x[i])/(x[i-1]+10) for i in 2:n])
```

```
0.0004852622754867377
```

Рис. 60: Вектор: вариант 1-2

- выберите элементы вектора □, значения которых больше 600, и выведите на экран; определите индексы этих элементов;

```
t = []
ind = []
for i in 1:n
    if y[i] > 600
        append!(t, y[i]); append!(ind, i)
    end
end
t
```

```
90-element Vector{Any}:
 695
 616
 625
 989
 897
 601
 949
 939
 764
 601
 868
 628
 636
   ⋮
 918
 877
 638
 823
 917
 992
 859
 666
 657
 987
 621
 748
```

Рис. 61: Вектор: вариант 1

```
ind
```

```
90-element Vector{Any}:
   3
   9
  10
  11
  13
  16
  24
  26
  27
  33
  34
  42
  43
   ⋮
 221
 222
 223
 224
 225
 226
 235
 237
 240
 241
 245
 247
```

Рис. 62: Вектор: вариант 1

```
t = y[findall(x->x>600, y)]
```

```
90-element Vector{Int64}:
 695
 616
 625
 989
 897
 601
 949
 939
 764
 601
 868
 628
 636
   ⋮
 918
 877
 638
 823
 917
 992
 859
 666
 657
 987
 621
 748
```

```
findall(x->x>600, y)
```

```
90-element Vector{Int64}:
  3
  9
 10
 11
 13
 16
 24
 26
 27
 33
```

Рис. 63: Вектор: вариант 2

- определите значения вектора □, соответствующие значениям вектора □, значения которых больше 600 (под соответствием понимается расположение на аналогичных индексных позициях);

```
hcat(ind, y[ind], x[ind])
```

```
90×3 Matrix{Any}:
   3  695  795
   9  616  469
  10  625  367
  11  989  187
  13  897   80
  16  601  678
  24  949  240
  26  939  783
  27  764   90
  33  601   16
  34  868  872
  42  628  354
  43  636  697
   ⋮
 221  918  169
 222  877  990
 223  638   24
 224  823  791
 225  917  908
 226  992  890
 235  859  920
 237  666  570
 240  657  994
 241  987  204
 245  621  613
 247  748  727
```

```
hcat(findall(temp->temp>600, y), y[findall(temp->temp>600, y)], x[findall(temp->temp>600, y)])
```

```
90×3 Matrix{Int64}:
   3  695  795
   9  616  469
  10  625  367
  11  989  187
  13  897   80
  16  601  678
  24  949  240
  26  939  783
  27  764   90
```

Рис. 64: Значение вектор: вариант 1-2

- сформируйте вектор $(|x_1 - \overline{x}|^{\frac{1}{2}}, |x_2 - \overline{x}|^{\frac{1}{2}}, \dots, |x_n - \overline{x}|^{\frac{1}{2}})$;

```
x_mean = 0
for i in x
    x_mean += i/n
end
t = [abs(i-x_mean)^(1/2) for i in x]
```

```
250-element Vector{Float64}:
  4.558947246898125
 17.15272573091519
 17.685700438489846
 18.660546615788082
  3.844996749023332
 19.1106253168231
 19.359338831685342
 16.43824808183646
  3.6353816856005654
 10.733871622112872
 17.181850889819756
 11.628241483560616
 20.055323482806255
  ⋮
 22.666803921153065
 22.622643523691035
 16.679808152373937
 16.10018633432545
 13.609408510291695
  9.528063811709071
 11.436083245587188
 10.187443251375685
 15.64557445413878
 19.073122450191526
 12.657645910673912
 13.12310938764133
```

Рис. 65: Вектор: вариант 1

```
t = abs.(x.-sum(x)/size(x)[1]).^(1/2)
```

```
250-element Vector{Float64}:
  4.558947246898125
 17.15272573091519
 17.685700438489846
 18.660546615788082
  3.844996749023332
 19.1106253168231
 19.359338831685342
 16.43824808183646
  3.6353816856005654
 10.733871622112872
 17.181850889819756
 11.628241483560616
 20.055323482806255
  ⋮
 22.666803921153065
 22.622643523691035
 16.679808152373937
 16.10018633432545
 13.609408510291695
  9.528063811709071
 11.436083245587188
 10.187443251375685
 15.64557445413878
 19.073122450191526
 12.657645910673912
 13.12310938764133
```

Рис. 66: Вектор: вариант 2

- определите, сколько элементов вектора отстоят от максимального значения не более, чем на 200;

```
# максимальное значение включено
ymax = 0
for i in y
    if i > ymax
        ymax = i
    end
end
counte = 0
for i in y
    if ymax-i <= 200
        counte += 1
    end
end
counte
```

```
53
```

```
ymax = maximum(y)
size(findall(x-> ymax-x<=200, y))[1]
```

```
53
```

Рис. 67: Вариант 1

```
# без включения максимального значения
ymax = 0
for i in y
    if i > ymax
        ymax = i
    end
end
counte = 0
for i in y
    if 0<ymax-i <= 200
        counte += 1
    end
end
counte
```

```
52
```

```
ymax = maximum(y)
size(findall(x-> 0<ymax-x<=200, y))[1]
```

```
52
```

Рис. 68: Вариант 2

- определите, сколько чётных и нечётных элементов вектора □;

```
counte = [0,0]
for i in x
    if i%2 == 0
        counte[1] += 1
    else
        counte[2] += 1
    end
end
counte[1], counte[2]
```

```
(130, 120)
```

```
size(findall(iseven, x))[1], size(findall(isodd, x))[1]
```

```
(130, 120)
```

Рис. 69: Вариант 1-2

- определите, сколько элементов вектора □ кратны 7;

```
counte = 0
for i in x
    if i % 7 == 0
        counte += 1
    end
end
counte
```

```
30
```

```
size(findall(x->x%7 == 0, x))[1]
```

```
30
```

Рис. 70: Вариант 1-2

- отсортируйте элементы вектора □ в порядке возрастания элементов вектора □;

```
num = trunc(Int, n/2)
orde = []
yordered = zeros(Int64, 2, num)
ycopy = copy(y)
for i in 1:num
    tm = [-1, 1000]
    imm = [0, 0]
    for j in 1:size(ycopy)[1]
        if ycopy[j]>tm[1]
            imm[1] = j
            tm[1] = ycopy[j]
        end
        if ycopy[j]<tm[2]
            imm[2] = j
            tm[2] = ycopy[j]
        end
    end
    yordered[1, i] = tm[2]; yordered[2, num-i+1] = tm[1]
    imm = sort(imm, rev=true)
    for k in imm
        ycopy = deleteat!(ycopy, k)
    end
end
yordered = vcat(yordered[1, :], yordered[2,:])
ycopy = copy(yordered)
while size(ycopy)[1]>0
    temp = ycopy[1]
    if size(ycopy)[1]>1
        while ycopy[2] == temp
            deleteat!(ycopy,2)
        end
    end
    append!(orde, findall(isequal(temp), y))
    deleteat!(ycopy, 1)
end
hcat(yordered, y[orde], x[orde])
```

```
250×3 Matrix{Int64}:
   3    3  634
   4    4  347
   5    5  707
  15   15  297
  26   26  136
  28   28  176
```

Рис. 71: Вариант 1

```
orde = sortperm(y)
hcat(sort(y), y[orde], x[orde])
```

```
250×3 Matrix{Int64}:
   3    3  634
   4    4  347
   5    5  707
  15   15  297
  26   26  136
  28   28  176
  32   32   90
  40   40  200
  41   41  411
  43   43   58
  44   44  333
  48   48  906
  53   53  626
   ⋮
 953  953  216
 955  955  577
 962  962  238
 965  965  622
 974  974  608
 974  974  576
 976  976  747
 978  978  573
 987  987  204
 989  989  187
 992  992  890
 999  999  220
```

Рис. 72: Вариант 2

- выведите элементы вектора □, которые входят в десятку наибольших (top-10);

```
# Могут быть повторяющиеся значения
t = []
xstr = join(x, "-")
while size(t)[1] < 10
    temp = 0
    for j in 1:size(x)[1]
        if x[j] >= temp && !(x[j] in t)
            temp = x[j]
        end
    end
    for j in 1:count(string(temp), xstr)
        append!(t, temp)
    end
end
t = t[1:10]
```

```
10-element Vector{Any}:
 996
 996
 994
 990
 990
 986
 981
 979
 965
 965
```

```
t = sort(x, rev=true)[1:10]
```

```
10-element Vector{Int64}:
 996
 996
 994
 990
 990
 986
 981
 979
 965
 965
```

Рис. 73: Вариант 1

```
# Без повторяющихся значений
t = []
xstr = join(x, "-")
while size(t)[1] < 10
    temp = 0
    for j in 1:size(x)[1]
        if x[j] >= temp && !(x[j] in t)
            temp = x[j]
        end
    end
    append!(t, temp)
end
t
```

```
10-element Vector{Any}:
 996
 994
 990
 986
 981
 979
 965
 962
 957
 956
```

```
t = unique(sort(x, rev=true))[1:10]
```

```
10-element Vector{Int64}:
 996
 994
 990
 986
 981
 979
 965
 962
 957
 956
```

Рис. 74: Вариант 2

- сформируйте вектор, содержащий только уникальные (неповторяющиеся) элементы вектора x;

```
x_unique = []
for i in x
    if !(i in x_unique)
        append!(x_unique, i)
    end
end
x_unique
```

```
220-element Vector{Any}:
 503
 188
 795
 134
 497
 117
 857
 212
 469
 367
 187
 347
  80
   ⋮
 153
 570
 735
 994
 204
 223
 613
 586
 727
 846
 322
 310
```

```
unique(x)
```

```
220-element Vector{Int64}:
 503
 188
 795
 134
 497
```

Рис. 75: Вариант 1-2

17. Выполним 4 задание: создадим массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100.

```
squares = [i^2 for i in 1:100]
```

```
100-element Vector{Int64}:
     1
     4
     9
    16
    25
    36
    49
    64
    81
   100
   121
   144
   169
     ⋮
  7921
  8100
  8281
  8464
  8649
  8836
  9025
  9216
  9409
  9604
  9801
 10000
```

Рис. 76: Задание 4

18. Подключим пакет Primes (функции для вычисления простых чисел).

```
import Pkg; Pkg.add("Primes")
```

```
    Updating registry at `C:\Users\User\.julia\registries\General.toml`
   Resolving package versions...
   Installed IntegerMathUtils ─ v0.1.2
   Installed Primes ──────────── v0.5.6
    Updating `C:\Users\User\.julia\environments\v1.10\Project.toml`
  [27ebfcd6] + Primes v0.5.6
    Updating `C:\Users\User\.julia\environments\v1.10\Manifest.toml`
  [18e54dd8] + IntegerMathUtils v0.1.2
  [27ebfcd6] + Primes v0.5.6
Precompiling project...
  ✓ IntegerMathUtils
  ✓ Primes
  2 dependencies successfully precompiled in 26 seconds. 325 already precompile
d.
```

Рис. 77: Подключение пакета Primes

19. Выполним 5 задание: сгенерируйте массив myprimes, в котором будут храниться первые 168 простых чисел. Определите 89-е наименьшее простое число. Получите срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа.

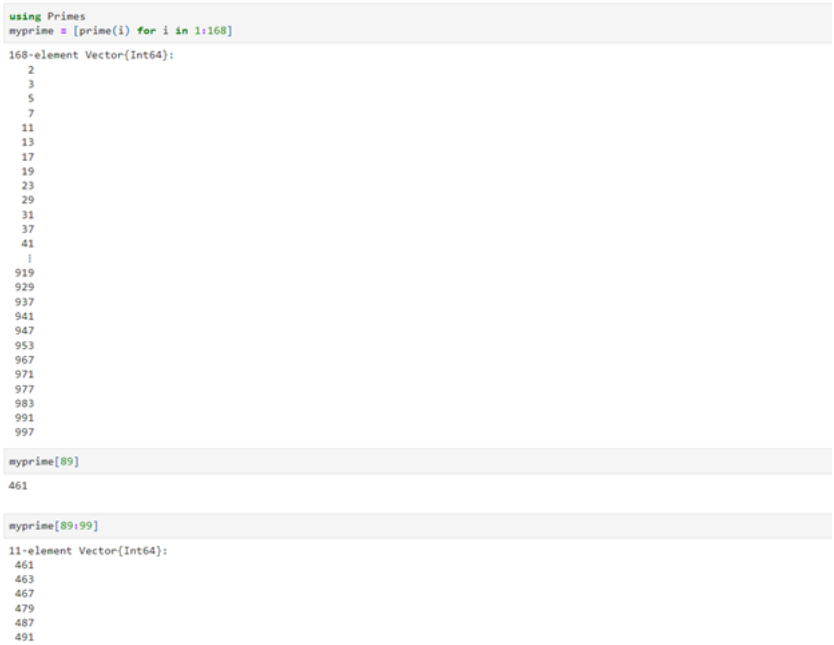

Рис. 78: Задание 5

20. Выполним 6 задание: вычислим следующие выражения:

- $\sum_{i=10}^{100}(i^3 + 4i^2)$;

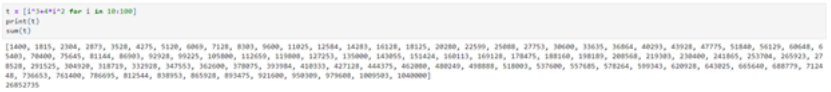

Рис. 79: Задание 6.1

- $\sum_{i=1}^{M}(\frac{2^i}{i} + \frac{3^i}{i^2})$, $M = 25$;

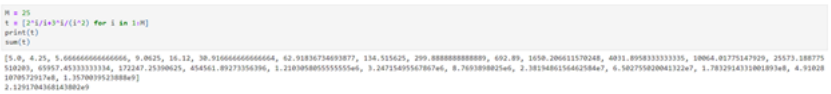

Рис. 80: Задание 6.2

- $1 + \frac{2}{3} + (\frac{2}{3}\frac{4}{5}) + ... + (\frac{2}{3}\frac{4}{5}...\frac{38}{39}) = 1 + \sum_{i=1}^{19}\prod_{j=1}^{i}\frac{2j}{2j+1}$

Рис. 81: Задание 6.3

Выводы по проделанной работе

Вывод

В результате выполнения работы мы изучили несколько структур данных, реализованных в Julia, и научились применять их и операции над ними для решения задач.

Были записаны скринкасты выполнения и защиты лабораторной работы.

Список литературы

[1] Julia: https://ru.wikipedia.org/wiki/Julia