Отчёт по лабораторной работе №2

Структуры данных

Студент: Кузнецова София Вадимовна

Содержание

Список иллюстраций

Список таблиц

Цель работы

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

Теоретическое введение

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.

Выполнение лабораторной работы

Кортежи

Кортеж (Tuple) — структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы). Синтаксис определения кортежа: (element1, element2, ...)

Примеры кортежей:

Рис. 0.1: Примеры кортежей

Примеры операций над кортежами:

```
[12]: # длина кортежа x2:
      length(x2)

[12]: 3

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

[14]: (1, 0.2, "tmp")

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

[16]: 5

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

[18]: (2, 3, 3)

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

[20]: (true, false)
```

Рис. 0.2: Примеры операция над кортежами

Словари

Словарь — неупорядоченный набор связанных между собой по ключу данных. Синтаксис определения словаря: Dict(key1 => value1, key2 => value2, ...)

Примеры словарей и операций над ними:

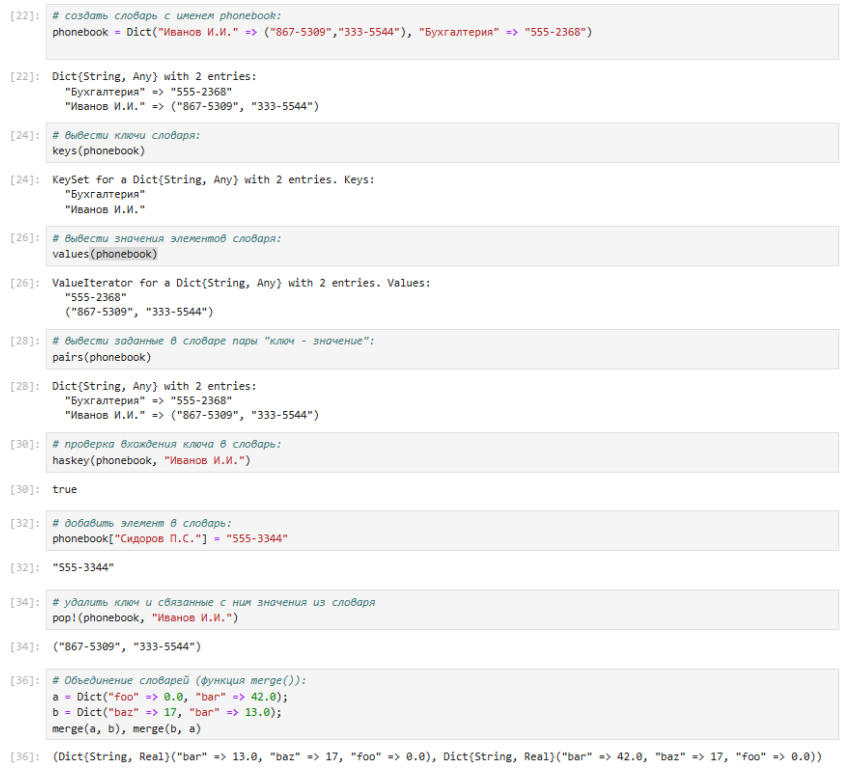

Рис. 0.3: Примеры словарей и операций над ними

Множества

Множество, как структура 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.

Примеры множеств и операций над ними:

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

[38]: Set{Int64} with 4 elements:
        5
        4
        3
        1

[40]: # создать множество из 11 символьных значений:
      B = Set("abrakadabra")

[40]: Set{Char} with 5 elements:
        'a'
        'd'
        'r'
        'k'
        'b'

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

[42]: false

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

[44]: true
```

Рис. 0.4: Примеры множеств и операций над ними

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

[46]: Set{Int64} with 4 elements:
        4
        2
        3
        1

[48]: # пересечение множеств:
      D = intersect(S1, S3)

[48]: Set{Int64} with 2 elements:
        2
        1

[50]: # разность множеств:
      E = setdiff(S3, S1)

[50]: Set{Int64} with 1 element:
        3

[52]: # проверка вхождения элементов одного множества в другое:
      issubset(S1, S4)

[52]: true

[54]: # добавление элемента в множество:
      push!(S4, 99)

[54]: Set{Int64} with 4 elements:
        2
        99
        3
        1

[56]: # удаление последнего элемента множества:
      pop!(S4)

[56]: 2
```

Рис. 0.5: Примеры множеств и операций над ними

Массивы

Массив — 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 . . .]

Примеры массивов:

```
[58]: # создание пустого массива с абстрактным типом:
      empty_array_1 = []

[58]: Any[]

[60]: # создание пустого массива с конкретным типом:
      empty_array_2 = (Int64)[]
      empty_array_3 = (Float64)[]

[60]: Float64[]

[62]: # вектор-столбец:
      a = [1, 2, 3]

[62]: 3-element Vector{Int64}:
       1
       2
       3

[64]: # вектор-строка:
      b = [1 2 3]

[64]: 1×3 Matrix{Int64}:
       1  2  3

[66]: # многомерные массивы (матрицы):
      A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]
      B = [[1 2 3]; [4 5 6]; [7 8 9]]

[66]: 3×3 Matrix{Int64}:
       1  2  3
       4  5  6
       7  8  9
```

Рис. 0.6: Примеры массивов

```
[82]: # одномерный массив из 8 элементов (массив $1 \times 8$)
      # со значениями, случайно распределёнными на интервале [0, 1):
      c = rand(1,8)

[82]: 1×8 Matrix{Float64}:
       0.538198  0.151628  0.349995  0.891214  …  0.272525  0.551418  0.792748

[86]: c = rand(2,3)

[86]: 2×3 Matrix{Float64}:
       0.736888  0.554953  0.54472
       0.345794  0.439748  0.0839246

[90]: # трёхмерный массив:
      D = rand(4, 3, 2)

[90]: 4×3×2 Array{Float64, 3}:
      [:, :, 1] =
       0.900014  0.459259  0.15483
       0.369843  0.55291   0.378339
       0.563482  0.104964  0.956263
       0.63803   0.029654  0.776925

      [:, :, 2] =
       0.839086   0.103351   0.168964
       0.91003    0.198215   0.950964
       0.444656   0.0881609  0.544898
       0.0677078  0.391478   0.0908424
```

Рис. 0.7: Примеры массивов

Примеры массивов, заданных некоторыми функциями через включение:

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

[92]: 10-element Vector{Float64}:
       1.0
       1.4142135623730951
       1.7320508075688772
       2.0
       2.23606797749979
       2.449489742783178
       2.6457513110645907
       2.8284271247461903
       3.0
       3.1622776601683795

[94]: # 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]

[94]: 5-element Vector{Int64}:
          3
         27
         75
        147
        243

[98]: # 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)]

[98]: 4-element Vector{Int64}:
        1
        9
       49
       81
```

Рис. 0.8: Примеры массивов, заданных некоторыми функциями через включение

Некоторые операции для работы с массивами:

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

[100]: 5-element Vector{Float64}:
        1.0
        1.0
        1.0
        1.0
        1.0

[102]: # двумерный массив 2x3 из единиц:
       ones(2,3)

[102]: 2×3 Matrix{Float64}:
        1.0  1.0  1.0
        1.0  1.0  1.0

[104]: # одномерный массив из 4 нулей:
       zeros(4)

[104]: 4-element Vector{Float64}:
        0.0
        0.0
        0.0
        0.0

[106]: # заполнить массив 3x2 цифрами 3.5
       fill(3.5, (3,2))

[106]: 3×2 Matrix{Float64}:
        3.5  3.5
        3.5  3.5
        3.5  3.5
```

Рис. 0.9: Некоторые операции для работы с массивами

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

```
[108]: 3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

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

```
[110]: 2×6 Matrix{Int64}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

```
[112]: # транспонирование
b'
```

```
[112]: 6×2 adjoint(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

Рис. 0.10: Некоторые операции для работы с массивами

```
[114]: # транспонирование
c = transpose(b)
```

```
[114]: 6×2 transpose(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

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

```
[116]: 10×5 Matrix{Int64}:
 19  19  19  11  15
 15  10  19  18  19
 20  13  16  10  15
 12  11  14  18  13
 12  12  10  10  20
 17  17  16  10  11
 18  10  19  10  20
 13  17  13  20  12
 18  11  12  17  14
 20  16  15  18  16
```

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

```
[118]: 10-element Vector{Int64}:
 19
 10
 13
 11
 12
 17
 10
 17
 11
 16
```

Рис. 0.11: Некоторые операции для работы с массивами

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

[120]: 10×2 Matrix{Int64}:
        19  15
        10  19
        13  15
        11  13
        12  20
        17  11
        10  20
        17  12
        11  14
        16  16

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

[122]: 10×3 Matrix{Int64}:
        19  19  11
        10  19  18
        13  16  10
        11  14  18
        12  10  10
        17  16  10
        10  19  10
        17  13  20
        11  12  17
        16  15  18

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

[124]: 3×2 Matrix{Int64}:
        15  19
        12  13
        17  11
```

Рис. 0.12: Некоторые операции для работы с массивами

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

[126]: 3-element Vector{Int64}:
        19
        11
        15

[128]: # сортировка по столбцам:
       sort(ar, dims=1)

[128]: 10×5 Matrix{Int64}:
        12  10  10  10  11
        12  10  12  10  12
        13  11  13  10  13
        15  11  14  10  14
        17  12  15  11  15
        18  13  16  17  15
        18  16  16  18  16
        19  17  19  18  19
        20  17  19  18  20
        20  19  19  20  20

[130]: # сортировка по строкам:
       sort(ar, dims=2)

[130]: 10×5 Matrix{Int64}:
        11  15  19  19  19
        10  15  18  19  19
        10  13  15  16  20
        11  12  13  14  18
        10  10  12  12  20
        10  11  16  17  17
        10  10  18  19  20
        12  13  13  17  20
        11  12  14  17  18
        15  16  16  18  20
```

Рис. 0.13: Некоторые операции для работы с массивами

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

```
[132]: 10×5 BitMatrix:
        1  1  1  0  1
        1  0  1  1  1
        1  0  1  0  1
        0  0  0  1  0
        0  0  0  0  1
        1  1  1  0  0
        1  0  1  0  1
        0  1  0  1  0
        1  0  0  1  0
        1  1  1  1  1
```

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

```
[134]: 28-element Vector{CartesianIndex{2}}:
        CartesianIndex(1, 1)
        CartesianIndex(2, 1)
        CartesianIndex(3, 1)
        CartesianIndex(6, 1)
        CartesianIndex(7, 1)
        CartesianIndex(9, 1)
        CartesianIndex(10, 1)
        CartesianIndex(1, 2)
        CartesianIndex(6, 2)
        CartesianIndex(8, 2)
        CartesianIndex(10, 2)
        CartesianIndex(1, 3)
        CartesianIndex(2, 3)
        ⋮
        CartesianIndex(10, 3)
        CartesianIndex(2, 4)
        CartesianIndex(4, 4)
        CartesianIndex(8, 4)
        CartesianIndex(9, 4)
        CartesianIndex(10, 4)
        CartesianIndex(1, 5)
        CartesianIndex(2, 5)
        CartesianIndex(3, 5)
        CartesianIndex(5, 5)
        CartesianIndex(7, 5)
        CartesianIndex(10, 5)
```

Рис. 0.14: Некоторые операции для работы с массивами

Самостоятельная работа

Выполнение заданий №1 и №2:

```
[136]: A = Set([0, 3, 4, 9])
       B = Set([1, 3, 4, 7])
       C = Set([0, 1, 2, 4, 7, 8, 9])
       P = union(intersect(A, B), intersect(A, C), intersect(B, C))
       println(P)
```

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

```
[138]: # Пример 1: множество строк
       set1 = Set(["cherry", "banana", "melon"])
       set2 = Set(["banana", "cherry", "data"])
       intersection = intersect(set1, set2)
       println(intersection)
```

```
Set(["cherry", "banana"])
```

```
[140]: # Пример 2: множество чисел
       set3 = Set([10, 20, 30])
       set4 = Set([20, 40, 50])
       difference = setdiff(set3, set4)
       println(difference)
```

```
Set([10, 30])
```

Рис. 0.15: Решение заданий №1 и №2

Выполнение задания №3(всех подпунктов):

```
[144]: N = 35
       array1 = collect(1:N)
       println(array1)

       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 3
       4, 35]

[146]: array2 = collect(N:-1:1)
       println(array2)

       [35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3,
       2, 1]

[148]: array3 = vcat(collect(1:N), collect(N-1:-1:1))
       println(array3)

       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 3
       4, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4,
       3, 2, 1]

[150]: tmp = [4, 6, 3]
       println(tmp)

       [4, 6, 3]
```

Рис. 0.16: Решение подпунктов задания №3

```
[152]: array4 = fill(tmp[1], 10)
       println(array4)

       [4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

[156]: array5 = repeat(tmp, 10)
       println(array5)

       [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, 6, 3, 4, 6, 3]

[158]: array6 = vcat(fill(tmp[1], 11), fill(tmp[2], 10), fill(tmp[3], 10))
       println(array6)

       [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

[160]: array7 = vcat(fill(tmp[1], 10), fill(tmp[2], 20), fill(tmp[3], 30))
       println(array7)

       [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Рис. 0.17: Решение подпунктов задания №3

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

       result_array = [2^tmp[i] for i in 1:3]
       result_array = vcat(result_array, repeat([2^tmp[3]], 4))

       result_string = join(result_array, "")
       count_6 = count(x -> x == '6', result_string)

       println("Результирующий массив: ", result_array)
       println("Количество цифры 6: ", count_6)

       Результирующий массив: [16, 64, 8, 8, 8, 8, 8]
       Количество цифры 6: 2

[166]: using Statistics

       x = 3:0.1:6
       y = [exp(x) * cos(x) for x in x]

       println("Среднее значение y: ", mean(y))

       Среднее значение y: 53.11374594642971
```

Рис. 0.18: Решение подпунктов задания №3

[168]:
```
xi = collect(3:3:36) * 0.1
yj = collect(1:3:34) * 0.2
vector11 = [(x, y) for x in xi, y in yj]
println(vector11)
```

```
[(0.30000000000000004, 0.2) (0.30000000000000004, 0.8) (0.30000000000000004, 1.4000000000000001) (0.30000000000000004, 2.0)
(0.30000000000000004, 2.6) (0.30000000000000004, 3.2) (0.30000000000000004, 3.8000000000000003) (0.30000000000000004, 4.4) (0.
30000000000000004, 5.0) (0.30000000000000004, 5.6000000000000005) (0.30000000000000004, 6.2) (0.30000000000000004, 6.800000000
000001); (0.6000000000000001, 0.2) (0.6000000000000001, 0.8) (0.6000000000000001, 1.4000000000000001) (0.6000000000000001, 2.
0) (0.6000000000000001, 2.6) (0.6000000000000001, 3.2) (0.6000000000000001, 3.8000000000000003) (0.6000000000000001, 4.4) (0.6
000000000000001, 5.0) (0.6000000000000001, 5.6000000000000005) (0.6000000000000001, 6.2) (0.6000000000000001, 6.800000000000
1); (0.9, 0.2) (0.9, 0.8) (0.9, 1.4000000000000001) (0.9, 2.0) (0.9, 2.6) (0.9, 3.2) (0.9, 3.8000000000000003) (0.9, 4.4) (0.
9, 5.0) (0.9, 5.6000000000000005) (0.9, 6.2) (0.9, 6.800000000000001); (1.2000000000000002, 0.2) (1.2000000000000002, 0.8) (1.
2000000000000002, 1.4000000000000001) (1.2000000000000002, 2.0) (1.2000000000000002, 2.6) (1.2000000000000002, 3.2) (1.2000000
000000002, 3.8000000000000003) (1.2000000000000002, 4.4) (1.2000000000000002, 5.0) (1.2000000000000002, 5.6000000000000005)
(1.2000000000000002, 6.2) (1.2000000000000002, 6.800000000000001); (1.5, 0.2) (1.5, 0.8) (1.5, 1.4000000000000001) (1.5, 2.0)
(1.5, 2.6) (1.5, 3.2) (1.5, 3.8000000000000003) (1.5, 4.4) (1.5, 5.0) (1.5, 5.6000000000000005) (1.5, 6.2) (1.5, 6.80000000000
0001); (1.8, 0.2) (1.8, 0.8) (1.8, 1.4000000000000001) (1.8, 2.0) (1.8, 2.6) (1.8, 3.2) (1.8, 3.8000000000000003) (1.8, 4.4)
(1.8, 5.0) (1.8, 5.6000000000000005) (1.8, 6.2) (1.8, 6.800000000000001); (2.1, 0.2) (2.1, 0.8) (2.1, 1.4000000000000001) (2.
1, 2.0) (2.1, 2.6) (2.1, 3.2) (2.1, 3.8000000000000003) (2.1, 4.4) (2.1, 5.0) (2.1, 5.6000000000000005) (2.1, 6.2) (2.1, 6.800
000000000001); (2.4000000000000004, 0.2) (2.4000000000000004, 0.8) (2.4000000000000004, 1.4000000000000001) (2.400000000000000
4, 2.0) (2.4000000000000004, 2.6) (2.4000000000000004, 3.2) (2.4000000000000004, 3.8000000000000003) (2.4000000000000004, 4.4)
(2.4000000000000004, 5.0) (2.4000000000000004, 5.6000000000000005) (2.4000000000000004, 6.2) (2.4000000000000004, 6.8000000000
00001); (2.7, 0.2) (2.7, 0.8) (2.7, 1.4000000000000001) (2.7, 2.0) (2.7, 2.6) (2.7, 3.2) (2.7, 3.8000000000000003) (2.7, 4.4)
(2.7, 5.0) (2.7, 5.6000000000000005) (2.7, 6.2) (2.7, 6.800000000000001); (3.0, 0.2) (3.0, 0.8) (3.0, 1.4000000000000001) (3.
0, 2.0) (3.0, 2.6) (3.0, 3.2) (3.0, 3.8000000000000003) (3.0, 4.4) (3.0, 5.0) (3.0, 5.6000000000000005) (3.0, 6.2) (3.0, 6.800
000000000001); (3.3000000000000003, 0.2) (3.3000000000000003, 0.8) (3.3000000000000003, 1.4000000000000001) (3.300000000000000
3, 2.0) (3.3000000000000003, 2.6) (3.3000000000000003, 3.2) (3.3000000000000003, 3.8000000000000003) (3.3000000000000003, 4.4)
(3.3000000000000003, 5.0) (3.3000000000000003, 5.6000000000000005) (3.3000000000000003, 6.2) (3.3000000000000003, 6.8000000000
00001); (3.6, 0.2) (3.6, 0.8) (3.6, 1.4000000000000001) (3.6, 2.0) (3.6, 2.6) (3.6, 3.2) (3.6, 3.8000000000000003) (3.6, 4.4)
(3.6, 5.0) (3.6, 5.6000000000000005) (3.6, 6.2) (3.6, 6.800000000000001)]
```

[172]:
```
M = 25
vector12 = [2^i / i for i in 1:M]
println(vector12)
```

```
[2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 102.4, 186.18181818
18182, 341.3333333333333, 630.1538461538462, 1170.2857142857142, 2184.5333333333333, 4096.0, 7710.117647058823, 14563.55555555
5555, 27594.105263157893, 52428.8, 99864.38095238095, 190650.18181818182, 364722.0869565217, 699050.6666666666, 1.34217728e6]
```

Рис. 0.19: Решение подпунктов задания №3

[185]:
```
N = 30
vector13 = ["fn$i" for i in 1:N]

println(vector13)
```

```
["fn1", "fn2", "fn3", "fn4", "fn5", "fn6", "fn7", "fn8", "fn9", "fn10", "fn11", "fn12", "fn13", "fn14", "fn15", "fn16", "fn1
7", "fn18", "fn19", "fn20", "fn21", "fn22", "fn23", "fn24", "fn25", "fn26", "fn27", "fn28", "fn29", "fn30"]
```

[189]:
```
using Random

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

filtered_y = y[y .> 600]
println("Элементы y > 600", filtered_y)

corresponding_x = x[findall(y .> 600)]
println("Соответствующие x: ", corresponding_x)

unique_x = unique(x)
println("Уникальные элементы x:", unique_x)
```

```
Элементы y > 600[806, 868, 990, 687, 853, 646, 870, 842, 774, 722, 874, 863, 809, 886, 793, 825, 695, 956, 679, 779, 650, 898,
803, 869, 817, 819, 964, 631, 836, 635, 845, 855, 801, 727, 935, 728, 645, 940, 918, 807, 674, 743, 750, 615, 735, 778, 762, 7
52, 863, 863, 718, 952, 735, 687, 826, 636, 973, 681, 969, 810, 706, 662, 833, 882, 796, 607, 909, 787, 777, 908, 827, 632, 78
3, 851, 794, 875, 989, 661, 813, 659, 939, 887, 642, 833, 930, 926, 936, 920, 619, 723, 660, 603, 723]
Соответствующие x: [524, 717, 191, 106, 181, 353, 193, 303, 968, 751, 375, 110, 588, 335, 213, 703, 570, 368, 740, 27, 22, 52,
383, 7, 129, 638, 962, 572, 933, 769, 586, 398, 870, 559, 372, 273, 627, 816, 513, 320, 370, 316, 415, 611, 266, 395, 700, 31
3, 83, 335, 890, 725, 451, 991, 6, 101, 798, 315, 314, 892, 124, 470, 456, 32, 175, 434, 675, 74, 46, 109, 232, 548, 876, 903,
296, 613, 21, 742, 305, 141, 926, 550, 814, 862, 233, 382, 148, 823, 435, 485, 774, 542, 100]
Уникальные элементы x:[524, 717, 597, 667, 739, 170, 753, 188, 191, 501, 610, 510, 45, 106, 600, 522, 683, 701, 589, 117, 181,
226, 223, 886, 353, 563, 537, 193, 699, 303, 920, 968, 976, 796, 751, 749, 387, 799, 63, 375, 932, 92, 859, 110, 937, 742, 36
9, 588, 405, 335, 763, 343, 213, 709, 150, 445, 87, 287, 281, 938, 703, 362, 570, 76, 368, 732, 740, 27, 22, 52, 383, 324, 69
2, 7, 217, 745, 458, 425, 957, 129, 598, 323, 638, 309, 962, 757, 572, 933, 949, 300, 37, 467, 777, 998, 769, 586, 870, 198, 9
29, 121, 407, 533, 461, 945, 398, 243, 338, 436, 439, 789, 139, 306, 559, 792, 532, 372, 289, 273, 627, 57, 772, 23, 977, 775,
979, 186, 816, 629, 513, 320, 421, 158, 851, 370, 316, 576, 443, 415, 942, 611, 590, 266, 395, 179, 700, 837, 313, 83, 890, 89
2, 442, 620, 725, 451, 706, 991, 297, 679, 6, 101, 798, 315, 727, 314, 124, 470, 456, 32, 175, 434, 621, 675, 13, 74, 214, 46,
109, 413, 232, 176, 548, 876, 746, 953, 903, 296, 225, 867, 448, 392, 613, 90, 21, 305, 715, 141, 566, 280, 926, 550, 414, 37
9, 814, 862, 233, 382, 377, 672, 148, 526, 823, 923, 435, 485, 584, 774, 542, 100, 804]
```

Рис. 0.20: Решение подпунктов задания №3

Выполнение задания №4:

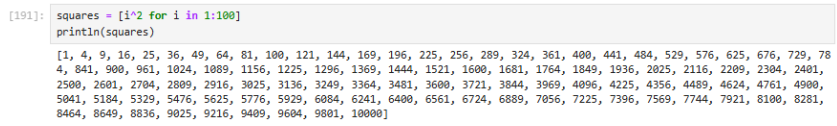

Рис. 0.21: Решение задания №4

Выполнение задания №5:

Рис. 0.22: Решение задания №5

Выполнение задания №6:

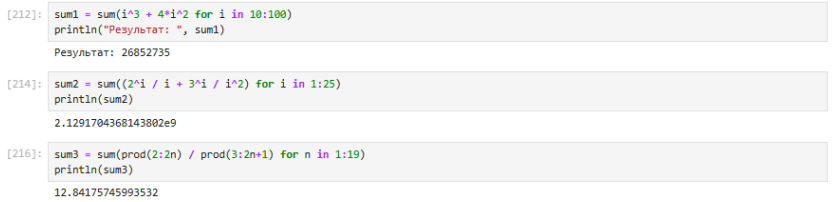

Рис. 0.23: Решение задания №6

Выводы

В ходе выполнения лабораторной работы были изучены несколько структур данных, реализованных в Julia, а также научились применять их и операции над ними для решения задач.

Список литературы

[1] Julia Documentation: https://docs.julialang.org/en/v1/