

Лабораторная работа №2. Компьютерный практикум по статистическому анализу данных

Леснухин Даниил Дмитриевич
Российский университет дружбы народов
Москва

Цель работы

Основная цель работы – изучить несколько структур данных, реализованных в Julia, научиться применять их и операции над ними для решения задач.

Задание

1. Используя Jupyter Lab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [[@julialang](#)]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia [[@juliadoc](#)].

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

Для начала выполним примеры из раздела про кортежи

The screenshot shows a Jupyter Notebook with a sidebar titled "СЛОВАРИ". The notebook contains the following code examples:

- [37] # создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => "")
- [38] # вывести значения элементов словаря:
values(phonebook)
- [39] # вывести значения элементов словаря:
values(phonebook)
- [40] # вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)
- [41] # проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")
добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"
- [42] # удалить ключ и связанные с ним значения из словаря
pop!(phonebook, "Иванов И.И.")
- [43] # 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)

Теперь выполним примеры из раздела про словари

```

▶ # 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) 

# 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]

```

Рис. 1: Примеры из раздела кортежи

МНОЖЕСТВА

```
[ин] 0 сек.
# создать множество из четырех целочисленных значений:
A = Set([1, 3, 4, 5])

...
Set[Int64] with 4 elements:
5
4
3
1

[ин] 0 сек.
# создать множество из 11 символьных значений:
B = Set("абракадабра")

...
Set[Char] with 5 elements:
'a'
'd'
'e'
'k'
'b'

[ин] 0 сек.
# проверка эквивалентности двух множеств:
S1 = Set([1,2]);
S2 = Set([3,4]);
IsEquivalent(S1,S2)
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
IsEquivalent(S3,S4)

...
true

[ин] 0 сек.
# объединение множеств:
C = Union(S1,S2)

...
Set[Int64] with 4 elements:
4
2
3
1

[ин] 0 сек.
# пересечение множеств:
D = Intersect(S1,S3)

...
Set[Int64] with 2 elements:
2
1

[ин] 0 сек.
# разность множеств:
E = SetDiff(S3,S1)

...
Set[Int64] with 1 element:
3

[ин] 0 сек.
# проверка вхождения элементов одного множества в другое:
IsSubset(S1,S4)

...
true

[ин] 0 сек.
# добавление элемента в множество:
push!(S4, 99)

...
Set[Int64] with 4 elements:
2
99
3
1

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

...
2
```

Выполним примеры из раздела про множества

Выполним примеры из раздела про массивы

Теперь перейдем к выполнению заданий.

Задание №1

Даны множества: $A = \{0, 3, 4, 9\}$, $B = \{1, 3, 4, 7\}$, $C = \{0, 1, 2, 4, 7, 8, 9\}$. Найдем $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$

Задание №2

Приведем свои примеры с выполнением операций над множествами элементов разных типов

▼ МАССИВЫ

```
[59] 0 сек.
    # создание пустого массива с абстрактным типом:
    empty_array_1 = []

    Any[]

[61] 0 сек.
    # создание пустого массива с конкретным типом:
    empty_array_2 = (Int64)[]
    empty_array_3 = (Float64)[]

    Float64[]

[62] 0 сек.
    # вектор-столбец:
    a = [1, 2, 3]

    ... 3-element Vector{Int64}:
        1
        2
        3

[63] 0 сек.
    # вектор-строка:
    b = [1 2 3]

    1x3 Matrix{Int64}:
        1 2 3

[64] 0 сек.
    # многомерные массивы (матрицы):
    A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]
    B = [[1 2 3]; [4 5 6]; [7 8 9]]

    3x3 Matrix{Int64}:
        1 2 3
        4 5 6
        7 8 9

[65] 0 сек.
    # одномерный массив из 8 элементов (массив $1 \times 8$)
    # со значениями, случайно распределёнными на интервале [0, 1]:
    c = rand(1,8)

    1x8 Matrix{Float64}:
        0.0859245 0.479945 0.162709 0.0130608 ...
        0.646947 0.973238 0.871883

[71] 0 сек.
    # многомерный массив $2 \times 3$ (2 строки, 3 столбца) элементов
    # со значениями, случайно распределёнными на интервале [0, 1]:
    C = rand(2,3)

    2x3 Matrix{Float64}:
        0.178175 0.725235 0.0246914
        0.174706 0.126942 0.565709
```

Рис. 2: Примеры использования массивов

```

[72]
0
сек.

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

    ... 4x3x2 Array{Float64, 3}:
[:, :, 1] =
0.667413 0.327224 0.995605
0.670233 0.866103 0.894114
0.473628 0.707488 0.201904
0.677968 0.270496 0.652811

[:, :, 2] =
0.8493652 0.419159 0.194344
0.575047 0.94959 0.493161
0.625627 0.935972 0.0933416
0.486154 0.958258 0.0225988

[73]
0
сек.

    # 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)

[74]
0
сек.

ar_1 = [3*i^2 for i in 1:2:9]

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

[75]
0
сек.

    # 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. 3: Примеры использования массивов

```

76] # одномерный массив из пяти единиц:
0
ones(5)

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

77] # двумерный массив 2x3 из единиц:
0
ones(2,3)

... 2x3 Matrix{Float64}:
1.0 1.0 1.0
1.0 1.0 1.0

78] # одномерный массив из 4 нулей:
0
zeros(4)

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

79] # заполнить массив 3x2 цифрами 3.5
0
fill(3.5,(3,2))

3x2 Matrix{Float64}:
3.5 3.5
3.5 3.5
3.5 3.5

80] # заполнение массива посредством функции repeat():
0
repeat([1,2],3,3)

... 3x6 Matrix{Int64}:
1 2 1 2 1 2
1 2 1 2 1 2
1 2 1 2 1 2

81] # преобразование одномерного массива из целых чисел от 1 до 12
0
# в двумерный массив 2x6
a = collect(1:12)
b = reshape(a,(2,6))

2x6 Matrix{Int64}:
1 3 5 7 9 11
2 4 6 8 10 12

82] # транспонирование
0
b'
# транспонирование
c = transpose(b)

```

Рис. 4: Примеры использования массивов

```

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

*** 10x5 Matrix{Int64}:
15 19 15 18 11
15 16 13 19 20
13 19 12 11 14
17 20 14 11 12
13 16 17 17 17
11 17 12 13 19
13 14 17 16 10
17 20 19 15 10
15 20 19 14 19
13 18 11 16 16

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

10-element Vector{Int64}:
19
16
19
20
16
17
14
20
20
18

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

10x2 Matrix{Int64}:
19 11
16 20
19 14
20 12
16 17
17 19
14 10
20 10
20 19
18 16

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

10x3 Matrix{Int64}:
19 15 18
16 13 19
19 12 11
20 14 11
16 17 17
17 12 13
14 17 16
20 19 15
20 19 14
18 11 16

```

Рис. 5: Примеры использования массивов

```

[88] 0 сек.
✓ // значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]

[89] 0 сек.
▼ 3x2 Matrix{Int64}:
15 20
17 12
11 19

[90] 0 сек.
► // значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]

[91] 0 сек.
▼ ... 3-element Vector{Int64}:
15
18
11

[92] 0 сек.
► // сортировка по столбцам:
sort(ar,dims=1)

[93] 0 сек.
▼ ... 10x5 Matrix{Int64}:
11 14 11 11 10
13 16 12 11 10
13 16 12 13 11
13 17 13 14 12
13 18 14 15 14
15 19 15 16 16
15 19 17 16 17
15 20 17 17 19
17 20 19 18 19
17 20 19 19 20

[94] 0 сек.
► // поэлементное сравнение с числом
// (результат - массив логических значений):
ar .> 14

[95] 0 сек.
▼ ... 10x5 BitMatrix:
1 1 1 1 0
1 1 0 1 1
0 1 0 0 0
1 1 0 0 0
0 1 1 1 1
0 1 0 0 1
0 0 1 1 0
1 1 1 1 0
1 1 1 0 1
0 1 0 1 1

```

// возврат индексов элементов массива, удовлетворяющих условию:

```

findall(ar .> 14)

[96] 0 сек.
▼ ... 30-element Vector{CartesianIndex{2}}:
CartesianIndex(1, 1)
CartesianIndex(2, 1)
CartesianIndex(4, 1)
CartesianIndex(8, 1)
CartesianIndex(9, 1)
CartesianIndex(1, 2)
CartesianIndex(2, 2)
CartesianIndex(3, 2)
CartesianIndex(4, 2)
CartesianIndex(5, 2)
CartesianIndex(6, 2)
CartesianIndex(8, 2)

```

Рис. 6: Примеры использования массивов

Рис. 7: Задание №1. Работа с множествами

Задание №3

Создадим массивы разными способами, используя циклы

```

2

[109]
✓ 0 сек.
    Set1 = Set([1, 2, 3, "Hello"])

    Set{Any} with 4 elements:
      2
      "Hello"
      3
      1

[110]
✓ 0 сек.
    println("\nElements of set1:")
    for i in Set1
      println(i)
    end

    ...
    Elements of set1:
      2
      Hello
      3
      1

[111]
✓ 0 сек.
    print(in("Hello", Set1))

    true

[112]
✓ 1 сек.
    Set1 = push!(Set1, "World!")
    println("\nSet after adding one element:\n")

    for i in 1:5
      push!(Set1, i)
    end
    println("\nset after adding range of elements:\n", Set1)

    ...
    Set after adding one element:

    set after adding range of elements:
    Set(Any[5, 4, "World!", 2, "Hello", 3, 1])

```

Рис. 8: Задание №2. Примеры операций над множествами элементов разных типов

```

3.10
[122]
✓ 0 sek.
▶ using Statistics
y(x) = exp(x) * cos(x)
Y = [y(x) for x in 3:0.1:6]
mean(Y)

...
... 53.11374594642971

3.11
[125]
✓ 0 sek.
vect = []
for j in 1:3:34
    i = j^2
    push!(vect, ((0.1)^i, (0.2)^j))
end
println(vect)

...
Any[(0.0100000000000002, 0.2), (1.000000000000005e-8, 0.001600000000000003), (1.000000000000008e-14, 1.280000000000005e-5), (1.000000000000001e-20, 1.024000000000000e-66)

[126]
✓ 0 sek.
▶ vect_0 = [(2^i)/i for i in 1:25]
println(vect_0)

...
... [2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.88888888888886, 102.4, 186.18181818182, 341.333333333333, 630.1538461538462, 117

3.12
✓ 0 sek.
▶ vect_1 = []
for i in 1:30
    push!(vect_1, "fn$i")
end
println(vect_1)

...
Any["fn1", "fn2", "fn3", "fn4", "fn5", "fn6", "fn7", "fn8", "fn9", "fn10", "fn11", "fn12", "fn13", "fn14", "fn15", "fn16", "fn17", "fn18", "fn19", "fn20", "fn21", "fn22", "fn

3.13
✓ 0 sek.
▶ vect_x = []
for i in 1:250
    push!(vect_x, rand(0:999))
end
println(vect_x)
vect_y = []
for i in 1:250
    push!(vect_y, rand(0:999))
end
println(vect_y)

...
Any[901, 816, 949, 163, 559, 49, 598, 272, 911, 631, 73, 870, 436, 604, 508, 524, 501, 463, 626, 359, 352, 50, 326, 827, 150, 822, 319, 147, 380, 703, 722, 643, 964, 23, 430, Any[675, 500, 956, 77, 994, 598, 623, 756, 341, 523, 340, 148, 143, 933, 160, 759, 346, 411, 39, 811, 34, 593, 959, 842, 467, 313, 122, 115, 683, 694, 599, 768, 803, 617, 926

...
... -401;140;-872;831;39;574;158;69;-388;-291;75;-727;497;-444;251;-178;-90;-424;185;-325;241;909;516;-360;163;-700;-204;536;314;-104;46;160;-347;897;369;-519;-562;-193;-12;-227;

```

Задание №4

Создадим массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100

```

squares = [(i)^2 for i in 1:100]
print(squares)

...
... [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296,

```

Рис. 9: Задание №4.

Задание №5

Подключим пакет Primes (функции для вычисления простых чисел). Сгенерируем массив myprimes, в котором будут храниться первые 168 простых чисел. Определим 89-е наименьшее простое число. Получим срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа.

Задание №6

```
[135]
✓ 29
ex.
  import Pkg
  Pkg.add("Primes")

...
  Updating registry at `~/julia/registries/General.toml`
  Resolving package versions...
  Installed IntegerMathUtils - v0.1.3
  Installed Primes └── v0.5.7
    Updating `~/julia/environments/v1.11/Project.toml`
    [27ebfcfcd6] + Primes v0.5.7
    Updating `~/julia/environments/v1.11/Manifest.toml`
    [18e54dd8] + IntegerMathUtils v0.1.3
    [27ebfcfcd6] + Primes v0.5.7
  Precompiling project...
  3717.5 ms ✓ IntegerMathUtils
  756.6 ms ✓ Primes
  2 dependencies successfully precompiled in 11 seconds. 468 already precompiled.

[136]
✓ 0
ex.
  using Primes
  myPrimes = primes(prime(168))
  println(myPrimes)
  myPrimes[89]
  myPrimes[89:99]

...
  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
  11-element Vector{Int64}:
  461
  463
  467
  479
  487
  491
  499
  503
  509
  521
  523
```

Рис. 10: Задание №5. Работа с пакетом Primes

Вычислим следующие выражения

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

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

$$1 + \frac{2}{3} + \left(\frac{2}{3} \frac{4}{5} \right) + \left(\frac{2}{3} \frac{4}{5} \frac{6}{7} \right) + \dots + \left(\frac{2}{3} \frac{4}{5} \dots \frac{39}{39} \right).$$

```
[27]
0
ex.
  sum = 0
  for i in 10:100
    sum += i^3 + 4*i^2
  end
  println(sum)
...
  26852735

[2]
M = 25
sum = 0
for i in 1:M
  sum += 2^i/i + 3^i/i^2
end
println(sum)
...
  2.1291704368143882e9

[3]
N = 38
sum = 1
a_n = 1
for i in 2:N
  a_n *= 1/(i+1)
  sum += a_n
end
println(sum)
...
  6.976346137897618
```

Выводы

В результате выполнения данной лабораторной работы я изучил несколько структур данных, реализованных в Julia. Научился применять их и операции над ними для решения задач.