Лабораторная работа № 2

Структуры данных

Оразгелдиев Язгелди

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	18

Список иллюстраций

3.1	Примеры использования кортежей	7
3.2	Примеры использования кортежей	8
3.3	Примеры использования словарей	8
3.4	Примеры использования множеств	9
3.5	Примеры использования массивов	10
3.6	Примеры использования массивов	10
3.7	Примеры использования массивов	11
3.8	Задание №1. Работа с множествами	11
3.9	Задание №2. Примеры операций над множествами элементов раз-	
	ных типов	12
3.10	Задание №3. Работа с массивами	12
3.11	Задание №3. Работа с массивами	12
3.12	Задание №3. Работа с массивами	13
3.13	Задание №3. Работа с массивами	13
3.14	Задание №3. Работа с векторами	14
3.15	Задание №3. Работа с векторами	14
3.16	Задание №3. Работа с векторами	15
3.17	Задание №3. Работа с векторами	16
3.18	Задание №3. Работа с векторами	16
3.19	Задание №4, №5 и №6	17

Список таблиц

1 Цель работы

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

2 Задание

- 1. Используя Jupyter Lab, повторите примеры из раздела 2.2.
- 2. Выполните задания для самостоятельной работы

3 Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи. Кортеж (Tuple) – структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексиру- ются с единицы).

```
[1]: ()
[2]: $\text{if mapmax us annowmod nums String:} \\
[2]: $\text{favoritelang . ("Python", "Julia", "k")} \\
[2]: ("Python", "Julia", "k")
[3]: $\text{suppose us uponow us uponow nums.} \\
[3]: (1, 2, 3)
[4]: $\text{suppose us a annowmod paimox nums.} \\
[4]: (1, 2.0, "tmp")
[5]: $\text{uponom us annowmod paimox nums.} \\
[4]: (1, 2.0, "tmp")
[5]: $\text{uponom us annowmod paimox nums.} \\
[5]: (a - 2, b - 3)
```

Рисунок 3.1: Примеры использования кортежей

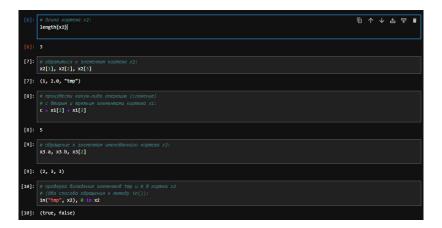


Рисунок 3.2: Примеры использования кортежей

Теперь выполним примеры из раздела про словари. Словарь – неупорядоченный набор связанных между собой по ключу данных.

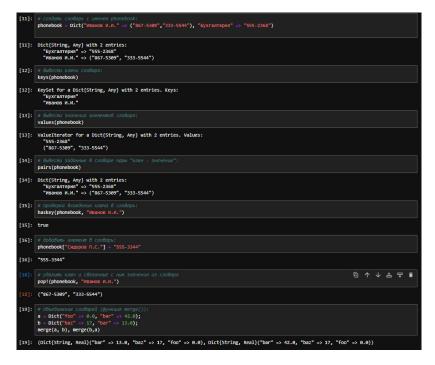


Рисунок 3.3: Примеры использования словарей

Выполним примеры из раздела про 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.

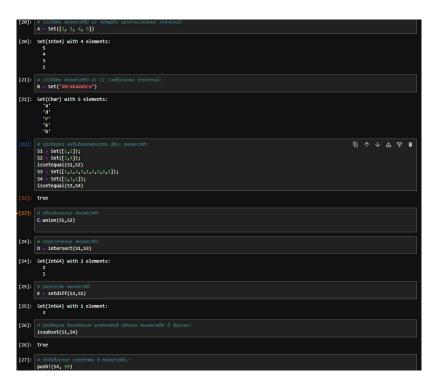


Рисунок 3.4: Примеры использования множеств

Выполним примеры из раздела про массивы. Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов.

Рисунок 3.5: Примеры использования массивов

Рисунок 3.6: Примеры использования массивов

Рисунок 3.7: Примеры использования массивов

Перейдем к выполнению заданий. 1. Даны множества: $\boxtimes = 0, 3, 4, 9, \boxtimes = 1, 3, 4, 7, \boxtimes = 0, 1, 2, 4, 7, 8, 9$. Найдем $\boxtimes = \boxtimes \cap \boxtimes \cup \boxtimes \cap \boxtimes \cup \boxtimes \cap \boxtimes \cup \boxtimes \cap \boxtimes$

```
: # Выполнение самостоятельной работы:

# Задание №1.

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)) # union(intersect(A, B), intersect(A, B)) = intersect(A, B)

println(P)

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

Рисунок 3.8: Задание №1. Работа с множествами

Приведем свои примеры с выполнением операций над множествами элементов разных типов

```
Set1 = Set([1,2,3, "Hello", "Geeks"])
println(Set1)
Set(Any[2, "Hello", "Geeks", 3, 1])

println("Nelsements of set:")
for i in Set1
    println(i)
end

Elements of set:
2
Hello
Geeks
3
1

println(in("Mello", Set1))
true

Set1 = push!(Set1, "welcome")
println("Nset after adding one element:\n", Set1)

Set after adding one element:
Set(Any["welcome", 2, "Hello", "Geeks", 3, 1])

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

Set after adding range of elements:
Set(Any["welcome", 2, "Hello", "Geeks", 3, 1])

Set after adding range of elements:\n", Set1)

Set after adding range of elements:\n", Set1)

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

Set after adding range of elements:
Set(Any[5, 4, "welcome", 2, "Hello", "Geeks", 3, 1])
```

Рисунок 3.9: Задание №2. Примеры операций над множествами элементов разных типов

Создадим массивы разными способами, используя циклы

Рисунок 3.10: Задание №3. Работа с массивами

```
tmp_deg: []
for i in 1:3
    push!(tmp_deg, 2^tmp[i])
end

print(vcnt(fill.(tmp_deg, [i, 1, 4])...))
count = 0

for i in tmp_deg
    if '6' in string(i)
        count = count = 1
    end
end
    println('\n', count)

[16, 64, 8, 8, 8, 8]
2

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

[Info: Precompiling Statistics [10745b16-79ce-11e8-11f9-7d13ad32a3b2]
53.11374594642971
```

Рисунок 3.11: Задание №3. Работа с массивами

```
** Thymnan 3.11. Beamop Guda (xi, yj), x = 0.1, i = 3, 6, 9, _ , 36, y = 0.2, j = 1, 4, 7, _ , 34.

**x_vector = 0.1 * collect(3:3:36)
**y_vector = 0.2 * collect(1:3:34)
**xy_pairs = [(x, y) for x in x_vector, y in y_vector]

**12x12 Matrix[Tuple[Float64, Float64]):
(0.3, 0.2) (0.3, 0.8) (0.3, 1.4) _ (0.3, 5.6) (0.3, 6.2) (0.3, 6.8)
(0.6, 0.2) (0.6, 0.8) (0.6, 1.4) (0.6, 5.6) (0.6, 6.2) (0.6, 6.8)
(1.2, 0.2) (1.2, 0.8) (1.2, 1.4) (1.2, 5.6) (1.2, 6.2) (1.2, 6.8)
(1.5, 0.2) (1.5, 0.8) (1.5, 1.4) (1.5, 5.6) (1.2, 6.2) (1.2, 6.8)
(1.5, 0.2) (1.8, 0.8) (1.8, 1.4) _ (1.8, 5.6) (1.8, 6.2) (1.8, 6.8)
(2.1, 0.2) (2.1, 0.8) (2.1, 1.4) (2.1, 5.6) (2.1, 6.2) (2.1, 6.8)
(2.2, 0.2) (2.2, 0.8) (2.1, 1.4) (2.7, 5.6) (2.7, 6.2) (2.7, 6.8)
(2.9, 0.2) (2.0, 0.8) (2.0, 1.4) (2.7, 5.6) (2.7, 6.2) (2.7, 6.8)
(3.0, 0.2) (3.0, 0.8) (3.0, 1.4) (3.0, 5.6) (3.0, 6.2) (3.0, 6.8)
(3.3, 0.2) (3.3, 0.8) (3.3, 1.4) (3.6, 5.6) (3.6, 6.2) (3.3, 6.8)
(3.3, 0.2) (3.3, 0.8) (3.5, 1.4) (3.6, 5.6) (3.6, 6.2) (3.6, 6.8)

**Inymax 3.12. Beamop c snememanu 2^(i)/i, i = 1, 2, _, N, N = 25.

**In = 25
**arr12 = [2^1 / i for i in 3:H]

**D5-element Vector(Float64):
2.0
2.6
2.6
2.8
8.888888888888866
102.4
18.6.1818181818182
341.33333333333
630.1538461538462
**120.0053140053140
```

Рисунок 3.12: Задание №3. Работа с массивами

```
# Nyukem 3.13. Bekemop Buda ("fn1", "fn2", ..., "fnN"), N = 38.

N = 38
arr13 = ["fn$i" for i in 1:N]

30-element Vector{String}:
    "fn1"
    "fn2"
    "fn3"
    "fn6"
    "fn7"
    "fn8"
    "fn9"
    "fn10"
    "fn11"
    "fn12"
    "fn13"
    i...
    "fn12"
    "fn13"
    i...
    "fn2"
    "fn30"
```

Рисунок 3.13: Задание №3. Работа с массивами

```
# Rynkm 3.14.

# Cosdoen Demops x u y:
n = 250
x = rand(0:999, n)
y = rand(0:999, n)
vec1 = y[2:end] - x[1:end-1] # Bekmop (y2 - x1, ..., yn - xn-1).

249-element Vector{Int64}:
285
-40
-16
507
-520
31
-498
865
342
439
213
-520
86
:
:
153
407
641
-495
490
504
575
-113
508
-362
674
-106
```

Рисунок 3.14: Задание №3. Работа с векторами

Рисунок 3.15: Задание №3. Работа с векторами

```
| Sum_expr = sum(exp.(-x[2:end]) .* x[1:end-1] .+ 10) # Bunucaenue cynnum.
| println(sum_expr)
| y_gt_600 = y[y .> 600] | solution |
```

Рисунок 3.16: Задание №3. Работа с векторами

```
count y within 200 = count(y -> y >= maximum(y) - 200, y) # Элементы y, отстоящие от максимума не более println(count_y_within_200)

even_count = count(iseven, x) # Количество четных и нечетных элементов в х.

odd count = count(ised, x)
println(even_count, odd_count)

multiples_of_7 = count(x -> x % 7 == 0, x) # Элементы x, кратные 7.

println(multiples_of_7)

250

sorted_x_by_y = x[sortperm(y)] # Copmupo8x x no возрастанию y.

250-element Vector(Int64):

172

129

884

386

74

762

801

22

644

423

...

928

939

922

934

987

921

100

748

790

152

557

445
```

Рисунок 3.17: Задание №3. Работа с векторами

Рисунок 3.18: Задание №3. Работа с векторами

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

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. Получии срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа

Рисунок 3.19: Задание №4, №5 и №6

4 Выводы

Мы изучили несколько структур данных, реализованных в Julia, научились применять их и операции над ними для решения задач.