Практика 1

1. Реализовать функцию, аналогичную встроенной функции reverse!, назвав её, например, reverse_user!, для следующих случаев: а) аргумент функции - вектор б) аргумент функции - матрица (2-мерный массив)

2. Аналогично, реализовать функцию, аналогичную встроенной функции сорудля следующих случаев: а) аргумент функции - вектор б) аргумент функции - матрица (2-мерный массив)

3. Реализовать алгоритм сортировки методом пузырька, написав следующие 4 обобщенные функции: bubblesort, bubblesort!, bubblesortperm, bubblesortperm!, по аналогии со встоенными функциями sort!, sort, sortperm!, sortperm, ограничившись только случаем, когда входной параметр есть одномерный массив (вектор).

4. На основе разработанных в пункте 1 функций, сотрирующих одномерный массив, написать соответствующие функции, которые бы могли получать на вход матрицу, и сортировать каждый из ее столбцов по отдельности. Имена функций оставить прежними, что были и в пункте 1, воспользовавшись механизмом множественной диспетчеризации языка Julia.

5. Написать функцию sortkey!(a, key_values), получающую на вход некоторый вектор a, и соответствующий вектор keyvalues ключевых значений элементов вектора a, осуществляющую сортировку вектора a по ключевым значениям его элементов, и возвращающую ссылку на вектор a. (Для сортировки вектора ключевых значений можно востпользоваться одной из разработанных в пункте 1 функций, или соответствующей встроенной функцией).

6. С использованием разработанной функции sortkey! написать функцию высшего порядка, с тем же именем sortkey!, но получающую на вход ключевую функцию и массив элементов некоторого типа, на множестве значений которых должна быть определена данная ключевая функция.

7. С использованием этой последней функции отсортировать столбцы какой-либо заданной числовой матрицы в порядке а) не убывания их сумм б) не убывания числа нулей в них

8. Написать функцию calcsort, реализующую сортировку методом подсчета числа значений. Рассмотреть 2 варианта функции (2 метода - в терминологии Julia): в первом варианте возможные значения элементов сортируемого массива задаются некоторым диапазоном, во втором - некоторым отсортированным массивом (вектором).

9. Применить эту разработанную функцию для сортировки столбцов матрицы по числу находящихся в них нулей (в каком случае сортировка подсчетом даст выигрыш по сравнению с любым другим методом сортировки?).

10. Написать функции insertsort!, insertsort, insertsortperm, insertsortperm! (по аналогии с пунктом 1) реализующие алгоритм сортировки вставками

11. Реализовать ранее написанную функцию insertsort! с помощью встроенной функции reduce (решение приведено в конце списка задач).

```
insertsort!(A)=reduce(1:length(A))do _, k # в данном случае при выполнении операции вставки  первый аргумент фуктически не используется
    while k>1 && A[k-1] > A[k]
        A[k-1], A[k] = A[k], A[k-1]
        k-=1
    end
    return A
end
```

12. Дополнить функцию insrtsort! процедурой "быстрого поиска".

13. Написать обобщенную функцию (nummax), получающую на вход итерируемый объект, содержащий некоторую последовательность (элементы которой можно сравнивать по величине), и возвращающую число максимумов этой последовательности.

14. Написать обобщенную функцию (findallmax), получающую на вход итерируемый объект, содержащий некоторую последовательность (элементы

которой можно сравнивать по величине), и возвращающую вектор, составленный из индексов элементов входной последовательности, имеющих максимальное значение.

15. Написать обобщенную функцию findallmax высшего порядка, получающую на вход итерируемый объект, содержащий некоторую последовательность и некоторую функцию (значение типа ::Function), и возвращающую вектор, составленный из индексов элементов входной последовательности, на которых заданная функция достигает максимального значения (речь и дет о сужении заданной функции на заданной последовательности).

Практика 2

1. Написать функцию sortkey(key_values, a), получающую на вход некоторый вектор a, и соответствующий вектор keyvalues ключевых значений элементов вектора a, осуществляющую сортировку вектора a по ключевым значениям его элементов, и возвращающую ссылку на вектор a. (Для сортировки вектора ключевых значений можно востпользоваться одной из разработанных в пункте 1 функций, или соответствующей встроенной функцией).

2. С использованием разработанной функции sortkey написать функцию высшего порядка, с тем же именем sortkey, но вместо массива ключевых значений получающую на вход ключевую функцию и массив элементов некоторого типа, на множестве значений которых должна быть определена данная ключевая функция.

3. С использованием этой последней функции отсортировать столбцы какой-либо заданной числовой матрицы в порядке а) не убывания их сумм б) не убывания числа нулей в них

4. Написать функцию calcsort, реализующую сортировку методом подсчета числа значений. Рассмотреть 2 варианта функции (2 её метода - в терминологии Julia): в первом варианте возможные значения элементов сортируемого массива задаются некоторым диапазоном, во втором - некоторым отсортированным массивом (вектором).

5. Применить эту разработанную функцию для сортировки столбцов матрицы по числу находящихся в них нулей (в каком случае сортировка подсчетом даст выигрыш по сравнению с любым другим методом сортировки?).

6. Написать функции insertsort!, insertsort, insertsortperm, insertsortperm! (по аналогии с пунктом 1) реализующие алгоритм сортировки вставками

7. Дополнить функцию insertsort! процедурой "быстрого поиска".

Практика 3-4

1. Задача 1. Написать функцию с заголовком findallmax(A::AbstractVector)::AbstractVector{Int}, возвращающую вектор индексов всех элементов массива A, имеющих максимальное значение. Алгоритм должен быть однопроходным, т.е. иметь асимптотическую оценку вычислительной сложности O(n).

2. Во-первых, можно запоминать индекс последнего элемента с левого конца массива, на котором закнчиватся гарантированный правильный порядок следования элементов массимва, с тем, чтобы на следующей итерации можно было бы начитнать сразу с этой позиции.

3. Во-вторых, пузырьковую сортировку можно сделать, так сказать, двунаправленной, т.е. сначала очередной наибольший элемент премещать до своего окончательного положения влево, а затем, начиная с предыдущего ему элемента перемещать очередной наименьший элемент до конца влево. Такой способ сортировки принято называть еще "шенкерной" сортрирвкой. Наглядно это показано, например, здесь. Задачи 2-3. Реализовать эти две разновидности пузырьковой сортировки.

4. Задача 4. Реализовать сортировку Шелла и проверить работоспособность алгоритма при следующих наборах значений промежутков (тут был LaTeX)

5. Задача 5. Написать функцию с заголовком => slice(A::AbstractVector, p::AbstractVector{<:Integer})

6. Пусть perm - это некоторый вектор перестановок индексов одномерного массива A. Написать свою реализацию встроенной функции permute!(A, perm), реализующую соответствующее премещение элементов массива A на месте (in-plice), т.е. без копирования их в новый массив . (Свой вариант этой функции можно назвать permute_!).

7. Реализовать встроенные функции вставки/добавления (deleteat!, insert!) элемента массива

8. Реализовать встроенные функции unique (возвращает новый массив, в который каждый элемент исходного массива входит только по одному разу), unique! (удаляет из исходного массива повторяющиеся элементы, оставляя каждый элемент в единственном экземпляре), allunique (проверяет, состоит ли данный массив только из уникальных элементов). Для функций unique и unique! обеспечить асимптотическую оценку вычислительной сложности O(n*log(n)).

9. Реализовать встроенную функцию reverse!, переставляющую элементы в обратном порядке в самом массиве, т.е. "на месте"

10. Написать функцию, осуществляющую циклический сдвиг массива на m позиций "на месте", т.е. без использования дополнительного массива.

11. Реализовать функцию, аналогичную встроенной функции transpose, с использованием вспомогательного массива

12. Реализовать функцию, аналогичную встроенной функции transpose, осуществляющую транспонирование матрицы "на месте" (без использования вспомогательного массива)

Практика 5

1. Аналогично построить индуктивную функцию и реализовать соответствующий программный код, вычисляющий значение второй производной многочлена в точке.

2. Сделать то же самое для 3-ей производной многочлена, заданного своими коэффициентами

3. Сделать то же самое для k-ой производной многочлена, заданного своими коэффициентами (здесь предполагается, что k - это параметр функции evaldiffpoly)

4. Определить функцицию с именем diff, для пользовательского типа Polynom, так чтобы ее можно было бы использовать так:

5. Реализовать функцию => divrem(a::AbstractVector{T},b::AbstractVector{T}) where {T <: Union{Rational, AbstractFloat}}`

6. Соответствующим образом переопределить операции % и ÷ (по аналогии с соответствующими целочисленными операциями) для пользовательского типа Polynom.

7. Для нашего пользовательского типа Polynom пределить ещё две функции, diff и integral, реализующие опрерации вифференцирования и интегрирования, соответственно.

8. Пусть задан массив данных a=[a[1],...,a[N]]. Написать функцию currenstd, получающую на вход некоторую последовательность, и возвращающую массив оценок "текущего" значения стандартного отклонения std (std=sqrt(D)), получаемых для по первым n членам последовательности a (n=1,2,...N). Данный алгоритм должен быть онопроходным.

9. Построить соответствующее индуктивное расширение реализовать однопроходный алгоритм, вычисляющий наибольшее значение обобщенной частичной суммы числовой последовательности.

10. Реализовать однопроходный алгоритм, возвращающий диапазон индексов элементов заданной числовой последовательности, соответствующего наибольшему значению обобщенной частичной суммы числовой последовательности.

Практика 6

1. Написать функцию, получающую 2 отсортированных массива A и B, и объединяющую их в одном отсортированном массиве C (length(C)=length(A)+length(B)=n). Алгоритм должен иметь оценку сложности O(n). Функцию можно назвать merge. Реализовать 2 варианта этой функции:
а) merge(A,B) - возвращает массив C;
Этот пункт надо реализовать двумя способами: в первом - для формировния массива C воспользоваться встроенными функциями push! и append!, предварительно инициализировав C значением пустого массива, а во втором - этими функциями не пользоваться, и сразу захватить под массив C необходимую для его размещения память (с помощью конструктора Vector{Type_vector}(undef, len_vector)).
б) merge!(A,B,C) - используется внешний массив C (поэтому в конце имени функции и поставлен восклицательный знак).

2. Написать функцию, выполняющую частичную сортировку. А именно, функция получает некоторый массив A и некоторое значение b, и переставляет элементы в массивае A так, что бы в нём сначала шли все элементы, меньшие b, затем - все, равные b, и затем, наконец, - все большие b. Алгоритм должен иметь оценку сложности O(n). Реализовать следующие 2 варианта этой функции:
а) с использованием 3-х вспомогательных массивов (с последующим их объединением в один);
б) без использования вспомогательного массива (все перемещения элементов должны осуществляться в переделах одного массива).

3. Написать функцию, выполняющую частичную сортировку. А именно, функция получает некоторый массив A и некотрое значение b, и переставляет элементы в массивае A так, чтобы в нём сначала шли все элементы, меньшие или равные b, а затем - все большие b. Алгоритм должен иметь оценку сложности O(n).

4. Написать функию, для заданного натурального числа n возвращающую массив всех биномиальных коэффициентов порядка n.

5. Написать функцию высшего порядка, для заданной функции f возвращающую значение её многочлена Бернштейна, заданного порядка n в заданной точке x.

Практика 7

1. Написать функцию pow(a, n::Integer), возвращающую значение a^n, и реализующую алгоритм быстрого возведения в степень.

2. Написать функцию fibonacci(n::Intrger), возвращающую n-ое число последовательности Фибоначчи, имеющую оценку алгоритмической сложности $O(log(n))$, и не используя известную формулу Бине.

3. Написать функцию log(a::Real,x::Real,ε::Real), реализующую приближенное вычисление логарифма по основанию a>1 числа x>0 с максимально допустимой погрешностью ε>0 (без использования разложения логарифмической функции в степенной ряд).

4. Написать функцию isprime(n)::Bool, возвращающую значение true, если аргумент есть простое число, и - значение false, в противном случае. При этом следует иметь ввиду, что число 1 простым не считается.

5. Написать функцию eratosphen(n), возвращающую вектор всех простых чисел, не превосходящих заданного натурального числа n.

6. Написать функцию factor(n), получающую некоторое натуральное число n, и возвращающую кортеж, состоящий из вектора его простых делителей (в порядке возрастания) и вектора кратностей этих делителей, т.е. выполняющую факторизацию заданного числа. Оценка вычислительной сложности алгоритма должна быть $O(\sqrt{n})$.

7. Написать фунуцию, получающую натуральный аргумент n, и возвращающую для него значение функции Эйлера.

8. Самостоятельно написать подобную функцию, реализующую расширенный алгоритм Евклида.

9. Написать функцию inv(m::Integer,n::Integer) возвращающий обратный элемент к значению m в кольце вычетов по модулю n (см. лекцию 3). При этом, если значение n не обратимо, то должно возвращаться значение nothing.

10. Написать функцию zerodivisors(m), возвращающую все делители нуля кольца вычетов по заданному модулю n.

11. Написать функцию nilpotents(n), для заданного n возвращающую диапазон (т.е. значение типа StepRange{Int64,Int64}), содержащий все не тривиальные нильпотенты кольца вычетов $\mathbb{Z}_n$. Возвращаемый диапазон должен быть пустым, если нетривиальных нильпотентов в кольце нет.

12. Написать функцию ord(a,p), возвращающую порядок заданного элемента a мультипликативной группы кольца вычетов по заданному простому модулю p.

13. Написать функцию высшего порядка bisection(f::Function, a, b; atol, rtol) реализующую решатель уравнения вида $f(x)=0$, реализующую метод деления отрезка пополам, где аргументы atol, rtol задают требуемую абсолютную и относительную точность (atol, rtol - это именованные параметры, которым можно присвоить также некоторые значения по умолчанию).

Практика 8

1. Реализовать рекурсивный вариант сортировки слияниями с оценками сложности и объема потребляемой памяти $O(n\cdot log(n))$.

2. Реализовать оба варианта нерекурсивной сортировки слияниями.

3. Сравнить время выполния различных реализаций алгоритма сортировки слияниями мужду собой и с встроенной функцией sort!. Длину массива следует взять достатоно большой, например, равной 1_000_000. Сортируемый массив формировать с помощью встроенного генератора случайных массивов randn (возвращающую массив заданного размера типа Float64, из нормально распределенных значений с нулевым математическим ожиданием (средним) и с единичным средним квадратическим отклонением).

4. Реализовать алгоритм быстрой сортировки Хоара двумя способами: основываясь на двух, указанных выше возможных вариантах частичной сортировки (partsort!, или - partsort2!).

5. Написать функцию, реализующую вычисление k-ой порядковой статистики, и - (на её основе) функции, вычисляющую медиану.

Практика 9

1. Задача 1. Написать функцию, вычисляющую n-ую частичную сумму ряда Тейлора функции $$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+...$$ для заданного значения аргумента x. Вычислительная сложность алгоритма должна иметь оценку $O(n)$.

2. Написать функцию, вычисляющую значение суммы ряда Тейлора функции $\cos(x)$ в заданной точке с машинной точностью.

3. Построить семейство графиков n-ых частичных сумм ряда Тейлора функции $\cos(x)$ на одном её периоде, для $n=2,4,8,16$.

4. Получить рекуррентные соотношения, требуемые для суммирования без повторных вычислений следующих степенных рядов

5. Следующий степенной ряд определяет семейство так называемых функций Бесселя 1-го рода порядка m ($m=0,1,2,...$) $$ J_m(x)=\Big(\frac{x}{2}\Big)^m\sum_{k=0}^{\infty}\frac{(-1)^k}{k!(k+m)!}\Big(\frac{x}{2}\Big)^{2k} $$ Написать функцию besselj(m,x), вычисляющую функцию Бесселя 1-го рода порядка m в точке $x \in \mathbb{R}$ с машинной точностью, и построить семейство графиков для $m=0,1,2,3,4,5$ (вид требуемых графиков см., например, здесь).

6. Написать функцию linsolve(A,b), получающую на вход невырожденную квадратную верхнетреугольную матрицу A (матрицу СЛАУ, приведенную к ступенчатому виду), вектор-столбец b (правую часть СЛАУ), и возвращающую решение соответствующей СЛАУ.

7. Написать функцию convert!(A), получающую на вход прямоугольную матрицу (например, - расширенную матрицу СЛАУ) и пробразующую эту матрицу к ступенчатому виду с помощью элементарных преобразований строк.

8. Написать функцию det(A), получающую на вход квадратную матрицу, и возвращающую значение её определителя.

9. Написать функцию inv(A), получающую на вход квадратную матрицу, и возвращающую обратную матрицу, если матрица обратима, или - значение nothing, в противном случае.

10. Написать функцию rang(A), получающую на вход матрицу (вообще говоря, прямоугольную), и возвращающую её ранг.

11. Написать функцию, получающую на вход матрицу СЛАУ приведенную к ступенчатому виду и возвращающую матрицу, содержащую фундаментальную систему решений этой СЛАУ, в случае, если система вырожденая, или, в противном случае, - пустой вектор-столбец (длина корого равна числу переменных, т.е. числу столбцов полученной матрицы).

12. Написать функцию, получающую на вход заранее приведенную к ступенчатому виду расширенную матрицу СЛАУ и возвращающую какое-либо частное решение этой системы, если оно существует, или значение nothing, в противном случае.

Практика 10

1. Реализовать метод Ньютона, написав функцию со следующим заголовком => newton(r::Function, x; ε_x=1e-8, ε_y=1e-8, nmaxiter=20)

2. Решить с помощью этой функции (newton) уравнение $\cos(x)=x$.
julia> newton(x->(x-cos(x))/(1+sin(x)), 0.5)

3. Реализовать ещё один метод функции newton со следующим заголовком newton(ff::Tuple{Function,Function}, x; ε_x=1e-8, ε_y=1e-8, nmaxiter=20)

4. Решить с помощью этого варианта функции newton уравнение $\cos(x)=x$.
julia> newton((x->x-cos(x), x->1+sin(x)), 0.5)

5. Реализовать еще один метод функции newton со следующим заголовком newton(ff, x; ε_x=1e-8, ε_y=1e-8, nmaxiter=20)

6. Решить с помощью последнего варианта функции newton уравнение $\cos(x)=x$.

7. Реализовать еще один метод функции newton со следующим заголовком newton(polynom_coeff::Vector{Number}, x; ε_x=1e-8, ε_y=1e-8, nmaxiter=20)

8. Разработать функцию, осуществляющую визуализацию проблемы Кэлли для заданного порядка n.

Практика 11

1. Написать функцию, возвращающу одномерный массив заданной длины, содержаий случайные точки плоскости типа Vector2D.

2. Написать функцию, которая получает на вход массив точек плоскости типа Point2D и отображает их на графике.

3. Написать функцию, получающую вектор кортежей, содержащих пары точек типа Vector2D, и возвращающую графический объект (типа Plots.Plot), содержащий изображение соответствующих отрезков, расположенных на плоскости.

4. Написать функцию, которая бы получала на вход аргумент segments, представляющий собой массив типа Vector{Tuple{Point2D,Point2D}}, или - генератор последовательности элементов типа Tuple{Point2D,Point2D} (представляющих некоторые отрезки), и возвращающую графический объект типа Plots.Plot, содержащий графики этих отрезков. Причем все точки пересечения этих отрезков толжна быть помечены красным крестообразным маркером.

5. Написать функцию, получающую на вход последовательность точек плоскости (их массив или генератор) и ещё пару (кортеж) точек плоскости, определяющих некоторую прямую. Функция должна вернуть графический объект типа Plots.Plot, содержащий график этих точек (в виде круглых маркеров) и график заданной прямой, причем все точки должны быть раскрашены в два цвета (синий и красный) таким образом, чтобы все точки лежащие по одну сторону от прямой были бы раскрашены в какой-то один цвет, и точки лежащие по разную сторону от прямой были бы разного цвета.

6. Задача 6. Написать функцию, получающую на вход последовательность точек плоскости (их массив или генератор) и ещё одну последовательность точек, определяющую координаты вершин некоторого многоугольника в порядке его обхода в одном из двух возможных направлений (например, для большей

определенности, - в положительном). Многоугольник не обязательно выпуклый, но без самопересечений сторон. Функция должна вернуть графический объект типа Plots.Plot, содержащий график этих точек (в виде круглых маркеров) и график заданного многугольника, причем все точки должны быть раскрашены так, чтобы все точки лежащие внутри многоугоугольника были бы красного цвета, а все точки лежащие снаружи - в синего.

7. Дана последовательность точек плоскости, определяющая вершины некоторго много угольника (в порядке их обхода в одном из двух возможных направлений). Требуется написать функцию, получающую на вход такую последовательность и возвращающую значение true, если многоугольник выпуклый, или значение false - в противном случае.

8. Написать функцию, получающую на вход последовательнсть точек плоскости и следующий набор параметров, определяющих эллипс: a, b - величины большой и малой полуосей эллипса, соответственно, (x_0, y_0) - координаты его центра, phi - угол поворота большой полуоси эллипса относительно положительного направления оси OX. Функция должна вернуть графический объект типа Plots.Plot, содержащий график этих точек (в виде круглых маркеров) и график заданного эллипса, причем все точки должны быть раскрашены так, чтобы все точки лежащие внутри эллипса были бы красного цвета, а все точки лежащие снаружи - в синего.

Практика 12

1. Написать функцию, получающую на вход вектор кортежей с координатами точек плоскости, и возвращающую вектор кортежей с координатами точек плоскости, являющимися вершинами выпуклой оболочки заданных точек. Реализовать метод Джарвиса. Выполнить графическое тестирование работы этой функции, т.е. для произвольно заданных исходных данных отображать на графике исходные точки и полученную выпуклую оболочку.

2. Написать функцию, получающую на вход вектор кортежей с координатами точек плоскости, и возвращающую вектор кортежей с координатами точек плоскости, являющимися вершинами выпуклой оболочки заданных точек. Реализовать метод Грехема. Выполнить графическое тестирование работы этой функции, т.е. для произвольно заданных исходных данных отображать на графике исходные точки и полученную выпуклую оболочку.

3. Написать функцию, получающую на вход вектор кортежей с координатами вершин некоторого плоского многоугольника, перечисленных в каком-то определенном, положительном или в отрицательном, направлении их обхода, и возвращающую значение его площади. Реализовать метод трапеций.

4. Написать функцию, получающую на вход вектор кортежей с координатами вершин некоторого плоского многоугольника, перечисленных в каком-то определенном, положительном или в отрицательном, направлении их обхода, и возвращающую значение его площади. Реализовать метод треугольников.

Практика 13-14

1. Определить тип, позволяющий итерировать все размещения с повторениями из n элементов по k.

Реализовать два способа:

a. на основе лексикографического порядка (как в лекции);

b. с использованем встроенной функции digits, возвращающей n-ичные цифры заданного целого числа; при этом реализовать функцию digits самомтоятельно (достаточно обеспечить только функциональность, необходимую для решения данной задачи).

При этом обеспечить возможность передавать в конструктор типа не только число n, но и, при желании, возможность передавать вместо него некоторое n-элементное множество (AbstractSet), с тем чтобы при итерировании получать наборы элементов непосредственно этого множества.

2. Определить тип, позволяющий итерировать все перестановки элементов заданного n элементного множества.

3. Определить тип, позволяющий итерировать все k-элементные подмножества заданного n-элементного множества.

4. Определить тип, позволяющий итерировать все размещения без повторений элементов заданного n элементного множества (в частности, последовательность {1,2,...,n} может задаваться просто числом n) по k.

5. Определить тип, позволяющий итерировать все разбиения заданного натурального числа n на положительные слагаемые. При этом, по-прежднему представляя разбиения как невозрастающие последовательности, перечислять их в порядке, обратном лексикографическому.

6. Дать решение всех остальных рассматривавшихся выше задач в функциональном стиле, т.е. с помощью разработки соответствующих функций, возвращающих тот или иной гененратор.

Практика 15

1. Написать функцию convert_to_nested(tree::ConnectList{T}, root::T) where T, получающую на вход дерево, представленное списком смежностей tree и индексом его корня root, и возвращающая представление того же дерева в виде вложенных векторов.

2. Написать функцию convert_to_list(tree::NestedVectors), получающую на вход дерево, представленное вложенными векторами, и возвращающая кортеж из списка смежностей типа ConnectList этого дерева и индекса его корня.

3. Написать функцию convert(tree::ConnectList{T}, root::T) where T, получающую на вход дерево, представленное списком смежностей tree и индексом его корня root, и возвращающая ссылку на связанные стркутруры типа Tree{T}, представляющие то же самое дерево, где

4. Написать функцию convert(tree::Tree{T}) where T, получающую на вход ссылку на связанные структуры типа Tree{T}, представляющие некоторое дерево, и возвращающая кортеж из списка смежностей типа ConnectList этого дерева и индекса его корня.

5. В лекции 7 были написаны следующие 5 функций:

- функция, возвращающая высоту заданного дерева
- функция, возвращающая число всех вершин заданного дерева
- функция, возвращающая число всех листьев заданного дерева
- функция, возвращающая наибольшую валентность по выходу вершин заданного дерева
- функция, возвращающая среднюю длину пути к вершинам заданного дерева

Во всех этих случаях дерево представлялось связанными структурами типа Tree{T}. Требуется дополнить эти определения функций новыми методами, в которых входной аргумент, через который каждая из функций получает данные, представляющие дерево, имел бы тип ConnectList, а также тип NestedVectors.

6. Написать функцию, получающую на вход имя некоторого типа (встоенного или пользовательского) языка Julia (тип этого аргумента - Type) и распечатывающая список всех дочерних типов в следующем формате:

```
ЗаданныйТип
    Подтип_1_певого_уровня
        Подтип_1_второго_уровня
            Подтип_1_1_третьего_уровня
            Подтип_1_2_третьего_уровня
            Подтип_1_3_третьего_уровня
            Подтип_1_4_третьего_уровня
        Подтип_2_второго_уровня
            Подтип_2_1_третьего_уровня
            Подтип_2_2_третьего_уровня
            Подтип_2_3_третьего_уровня
```

7. Требуется переписать эту функцию, используя в качестве исходных данных, представляющих дерево, список смежностей типа ConnectList и индекс его корня.

Практика 16

1. Написать функцию, получающую на вход весовую матрицу некоторого графа (в которой некоторым ребрам может быть приписан бесконечно большой вес), и возвращающую кортеж из числа, равного стоимости оптимального гамильтонова цикла, и вектор, в качестве своих элементов содержащий вектора с перестановками индексов вершин графа, соответствующих всем оптимальным гамильтоновым циклам.

2. Реализовать и протестировать алгоритм Форда-Беллмана.

3. Оценить асимптотическую сложность спроектированного алгоритма Форда-Беллмана (почему правильный ответ - $O(n^3)$?). Как можно было бы модернизировать разработанный алгоритм, чтобы его алгоритмическая сложность оценивалась как $O(n\cdot n_e)$, где n_e - число ребер (конечного веса) в графе? (подсказка: если в графе много фактически отсутствующих ребер, то для представления такого графа целесообразно воспользоваться разреженными массивами, см. лекцию 7, пункт 1.4).

4. Реализовать и протестировать алгоритм Флойда, включив в этот алгоритм проверку отсутствия циклов отрицательного суммарного веса.

5. Написать функцию floyd_next(G::Matrix), отличающуюся от определенной выше функции floyd(G::Matrix) тем, что она возвращает не только матрицу минимальных стоимостей "переездов" C, но и матрицу next, т.е. возвращает кортеж (C, next).

6. Написать функцию optpath_floyd(next::AbstractMatrix, i::Integer, j::Integer), которая бы по аналогии с функцией optpath_ford_bellman возвращала бы оптимальный путь, ведущий из заданной вершины i в заданную вершину j, если существует такой путь конечной стоимости, или - значение nothing, в противном случае.

7. Реализовать алгоритм Дейкстры, написав функцию dijkstra(G::AbstractMatrix, s::Integer), возвращающую вектор минимальных стоимостей путей из заданной вершины с индексом s во все остальные вершины.

Практика 17

1. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую вектор индексов его вершин, полученных в порядке поиска в глубину.

2. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую вектор индексов его вершин, полученных в порядке поиска в ширину.

3. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую вектор валентностей его вершин по выходу.

4. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую вектор валентностей его вершин по входу.

5. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую значение true, если он является сильно связным, и значение false - в противном случае.

6. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую значение true, если он является слабо связным, и значение false - в противном случае.

7. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую число компонент связности в неорграфе.

8. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую вектор, длина которого равна числу компонент связности, и каждый элемент которого содержит индекс какой-либо вершины из соответсвующей компоненты связности.

9. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа, и возвращающую true, если граф двудольный, и false - в противном случае.

10. Написать и протестировать функцию, получающую на вход список смежностей некоторого графа и индексы каких-либо двух его вершин, и

возвращающую кратчайший путь из первой вершины во вторую в виде вектора из индексов последовательности вершин, через которые проходит этот путь.

11. Написать и протестировать функцию, получающую на вход список смежностей некоторого орграфа, и возвращающую последовательность индексов его вершин в порядке, соответствующем топологической сортировке этого графа, или значение nothing, если граф содержит циклы.

Практика 18

1. Написать функцию, получающую на вход список смежностей некоторого ориентировнного, вообще говоря, эйлерова графа (мультиграфа), и возвращающую вектор, содержащий последовательность вершин в порядке обхода найденного эйлерова цикла.

2. Доработать решение задачи 1 таким образом, чтобы можно было снять требование сильной связности графа.

3. Доработать решение задачи 1 таким образом, чтобы в случае, если граф окажется не эйлеровым, в этом случае функция возвращала бы nothing.

4. Написать функцию, реализующую алгоритм Прима, с заголовком prime(G::Matrix), получающую на вход весовую матрицу некоторого связного простого неорграфа, и возвращающую вектор кортежей вида (v,u), представляющих собой ребра графа, составляющие остов, где u, v - индексы соответствующих вершин графа.

5. Написать функцию, реализующую алгоритм Краскала, с заголовком kruskal(G::Matrix), получающую на вход весовую матрицу некоторого связного простого неорграфа, и возвращающую вектор кортежей вида (v,u), представляющих собой ребра графа, составляющие остов, где u, v - индексы соответствующих вершин графа.