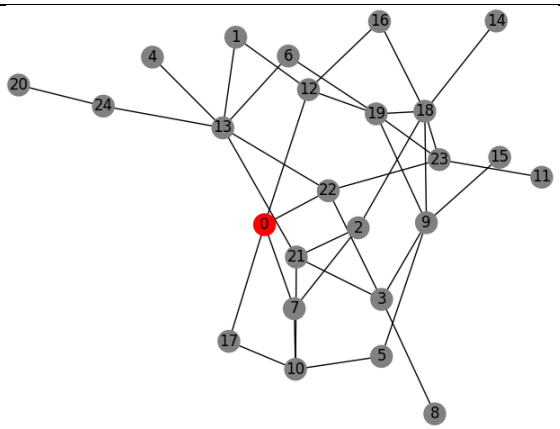
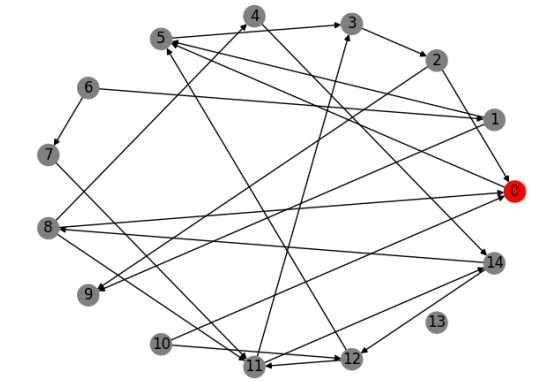
Реализация алгоритмов теории графов в Python

Имеется структура классов:

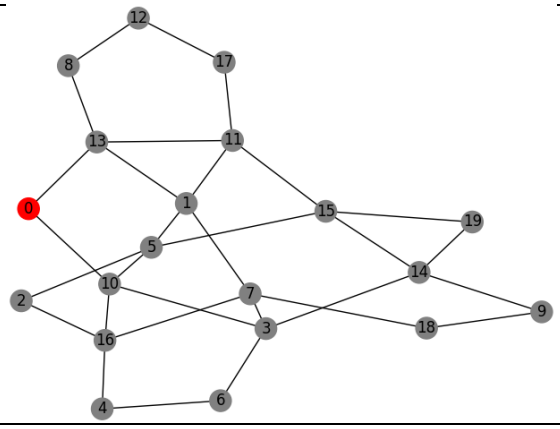
RandomGraph – класс неориентированных графов (родительский класс):
- случайные веса ребер
- случайные веса вершин
- только ненаправленные связи
- нет кратных ребер
- нет петель
- любые две вершины достижимы (связный граф)



RandomDiGraph – подкласс ориентированных графов:
- случайные веса ребер
- случайные веса вершин



RandomEulersGraph – подкласс неориентированных эйлеровых графов:
- единичные веса ребер
- единичные веса вершин



Экземпляры всех классов имеют следующие атрибуты:

Атрибут	Описание
n	Количество вершин графа. Тип – int.
W	Матрица весов графа. Тип – list of lists: W = [[0, 2, 6, 8, -, -, 3, -, -], [2, 0, 9, 3, -, 4, 9, -, -], [6, 9, 0, 7, -, -, -, -, -], [8, 3, 7, 0, 5, 5, -, -, -], [-, -, -, 5, 0, -, 8, 9, -], [-, 4, -, 5, -, 0, -, 6, 4], [3, 9,- , -, 8, -, 0, -, -], [-, -, -, -, 9, 6, -, 0, 1], [-, -, -, -, -, 4, -, 1, 0]] Пример:

W	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	2	6	8	-	-	3	-	-
1	2	0	9	3	-	4	9	-	-
2	6	9	0	7	-	-	-	-	-
3	8	3	7	0	5	5	-	-	-
4	-	-	-	5	0	-	8	9	-
5	-	4	-	5	-	0	-	6	4
6	3	9	-	-	8	-	0	-	-
7	-	-	-	-	9	6	-	0	1
8	-	-	-	-	-	4	-	1	0

A	Матрица смежности графа. Тип – list of lists: Пример: A | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** **0** | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 **1** | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 **2** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 **3** | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 **4** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 **5** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 **6** | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 **7** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 **8** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0
Al	Список смежности (список соседей). Тип – list of lists: Al = [[1,2,3,6], [0,2,3,5,6], [0,1,3], [0,1,2,4,5], [3,6,7], [1,3,7,8], [0,1,4], [4,5,8], [5,7]]
B	Матрица инциденций графа. Тип – list of lists.
R	Матрица достижимостей графа. Тип – list of lists.
D	Матрица кратчайших расстояний графа. Тип – list of lists: D = [[0, 2, 1], [2, 0, 3], [1, 3, 0]]
V	Матрица весов вершин. Тип – list.
E	Список ребер графа в формате (вес, начало, конец). Тип – list of tuples.
mean_degree	Средняя степень вершины графа (среднее количество ребер или дуг, выходящих из вершин). Тип – float.
min_edge_weight	Минимальный вес ребра. Тип – int. Значение по умолчанию – 1.
max_edge_weight	Максимальный вес ребра. Тип – int. Значение по умолчанию – 1.
min_node_weight	Минимальный вес вершины. Тип – float. Значение по умолчанию – 1.
max_node_weight	Максимальный вес вершины. Тип – float. Значение по умолчанию – 1.
num_of_edges	Количество ребер или дуг графа. Тип – int.
degrees	Список степеней вершин графа. Тип – list. Для ориентированного графа – список из двух списков.
dead_ends	Список висячих вершин графа. Тип – list.
isolated_nodes	Список изолированных вершин. Тип – list.
density	Плотность графа (0 – для пустого графа, 1 – для полного графа). Тип – float.
diameter	Диаметр графа. Тип – int.
is_connected	Метка связности графа. Тип – bool.

Классы имеют следующие реализованные методы:

class RandomGraph

__init__() – конструирует граф, создает атрибуты графа.
w_generation() – генерирует случайную матрицу весов W.
v_generation() – генерирует случайную матрицу весов вершин V.
replace_from_files() – считывает матрицы W и V из внешних файлов.
graph_draw() – рисует граф, выделяет путь на графе (при необходимости), выделяет дерево на графе (при необходимости), выделяет вершины графа (при необходимости). Для работы требуется установка модулей networkx, matplotlib.

class RandomDiGraph(RandomGraph)

w_generation() – генерирует случайную матрицу весов W ориентированного графа.
graph_draw() – рисует ориентированный граф

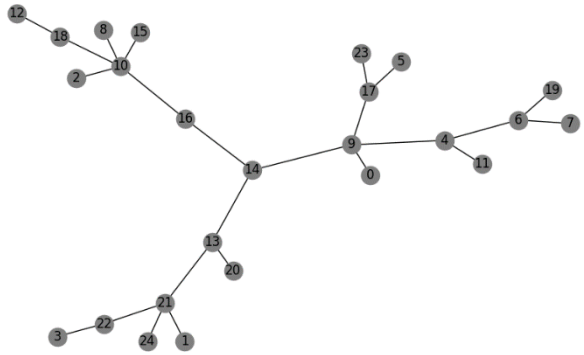
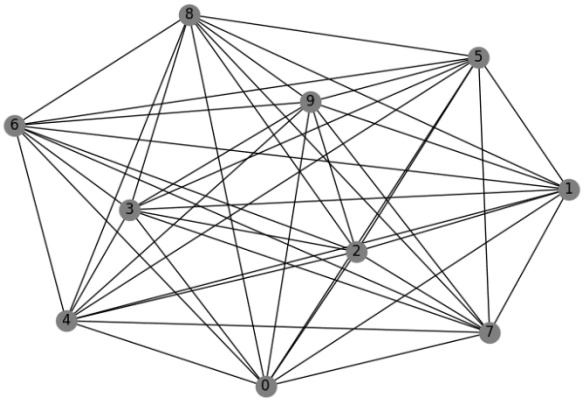
class RandomEulersGraph(RandomGraph)

w_generation() – генерирует случайную матрицу весов W эйлерового графа.

Как генерируется случайный граф класса RandomGraph.

Степень вершины d_i – количество ребер, инцидентных вершине

Средняя степень вершин ($\overline{d}$) – характеризует плотность ребер графа

Наименьший связный граф – дерево	Полный граф – содержит все ребра
	
$\boldsymbol{e = n - 1}$ ребро	$\boldsymbol{e = \frac{n \cdot (n-1)}{2}}$ ребер
$\boldsymbol{\overline{d} = \frac{2(n-1)}{n}}$	$\boldsymbol{\overline{d} = n - 1}$

При добавлении в граф одного ребра суммарная степень всех вершин увеличивается на 2.

$\boldsymbol{d_{\Sigma} = e \cdot 2}$ - суммарная степень всех вершин

$\boldsymbol{\overline{d} = \frac{2e}{n}}$ - средняя степень вершины

Если мы задаем среднюю степень вершины, то число ребер:

$$\boldsymbol{e = \frac{n \cdot \overline{d}}{2}}$$

Например:

$n = 20$

$\overline{d} = 3$

$e = 30$ ребер

Задание №1

Напишите и протестируйте методы класса RandomGraph и подкласса RandomDiGraph, которые выполняют следующие действия:

№	Описание метода	Имя метода
1	Метод класса RandomGraph. По матрице весов W вычисляет матрицу смежности A.	adjacency(self)
	Метод класса RandomGraph. По матрице смежности A вычисляет матрицу достижимостей R и проверяет, является ли граф связным.	reachability(self)
2	Метод классов RandomGraph и RandomDiGraph. По матрице весов *W* вычисляет матрицу инцидентности B	incidence(self)
	Метод классов RandomGraph и RandomDiGraph. По матрице весов W вычисляет список ребер E в формате (вес, начало, конец).	w_edges(self)
3	Метод класса RandomGraph. По матрице весов W вычисляет степени всех вершин, формирует список висячих вершин и список изолированных вершин.	node_degreeses(self)
	Метод класса RandomDiGraph. По матрице весов W вычисляет степени исхода и степени захода всех вершин, формирует список висячих вершин (с одной инцидентной дугой) и список изолированных вершин (без инцидентных дуг).	node_degreeses(self)
4	Метод класса RandomGraph. По матрице весов W и номеру вершины вычисляет список соседей вершины.	neighbors(self, v)
	Метод класса RandomGraph. По матрице весов W и номеру вершины вычисляет ближайшего соседа вершины	nearest_neighbor(self, v)
	Метод класса RandomGraph. По матрице весов W вычисляет список смежности Al.	adjacency_list(self)
5	Метод класса RandomGraph. По матрице весов W вычисляет матрицу кратчайших расстояний *D* (алгоритм Флойда-Уоршелла).	shortest_distances(self)
	Метод класса RandomGraph. По матрице кратчайших расстояний D вычисляет диаметр графа (расстояние между наиболее удаленными вершинами).	diameter_calc(self)
6	Метод классов RandomGraph и RandomDiGraph. По матрице весов W вычисляет метрики графа: число вершин, число ребер, наименьший вес ребра, наибольший вес ребра, наименьший вес вершины, наибольший вес вершины.	metrics(self)
	Статический метод классов RandomGraph и RandomDiGraph. В матрице весов W1 удаляет ребро edge при условии, что оно существует.	@staticmethod edge_deleter(W1, edge)
	Статический метод классов RandomGraph и RandomDiGraph. В матрице весов W1 добавляет ребро edge весом weight.	@staticmethod edge_inserter(W1, edge, weight)
	Статический метод класса RandomGraph. В матрице весов W1 удаляет ребра, инциндентные вершине node.	@staticmethod node_deleter(W1, node)
	Статический метод класса RandomGraph. В матрице весов W1 удаляет ребра, вес которых меньше заданного значения weight.	@staticmethod edge_filter(W1, weight)
7	Метод класса RandomGraph. По матрице весов W и списку вершин маршрута path вычисляет длину маршрута (False, если маршрута нет).	path_length(self, path)
	Метод класса RandomGraph. По матрице весов *W* и списку ребер edges_list вычисляет вес дерева, образованного ребрами (False, если хотя бы одного из ребер нет в графе).	tree_weight(self, edges_list)
8	Метод класса RandomGraph. В матрице весов W изменяет значения весов на отрицательные	negative_weights(self)
	Метод класса RandomGraph. Строит дополнение графа. Метод изменяет матрицу весов W на матрицу весов дополнения графа. Веса ребер принимаются =1.	inversion(self)

Задание №2

Напишите и протестируйте метод, выполняющий расчет по одному из алгоритмов теории графов.

Общая информация по исходным данным, требования к разрабатываемому методу

1. В задаче рассматриваются графы с заданными весами ребер (и при необходимости с заданными весами вершин). В некоторых задачах веса всех ребер могут равняться 1. Все графы являются простыми, то есть не допускают нескольких связей между вершинами. Размер графа до n=250.
2. Неориентированный граф создается как экземпляр класса RandomGraph, ориентированный граф – как экземпляр класса RandomDiGraph, эйлеров граф – как экземпляр класса RandomEulersGraph.
3. Необходимо предусмотреть отрисовку графа и результата, полученного при выполнении алгоритма (какой-либо путь, остовное дерево, центры графа). Для отрисовки графа использовать метод graph_draw().

Темы для разработки

№	Описание метода	Имя метода
1	**Поиск кратчайшего пути в графе по алгоритму Дейкстры** На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины На выходе: path – кратчайший маршрут между start и end (список вершин, входящих в маршрут) length – длина маршрута	dijkstra(self, start, end)
	Поиск кратчайшего пути между двумя вершинами графа по алгоритму Флойда-Уоршелла На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины На выходе: path – кратчайший маршрут между start и end (список вершин, входящих в маршрут) length – длина маршрута	floyd(self, start, end)
2	**Поиск k кратчайших путей в графе по алгоритму Йена** На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины k – количество путей На выходе: paths – кратчайшие маршруты между start и end (списки вершин, входящих в маршруты) length – длины маршрутов Примечание: для поиска кратчайших путей в графе используйте алгоритм Дейкстра	yen(self, start, end, k)

3	**Поиск кратчайшего пути между любой парой вершин графа по алгоритму Форда-Беллмана** На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины На выходе: path – кратчайший маршрут между start и end (список вершин, входящих в маршрут) length – длина маршрута	ford_bellman(self, start, end)
	Поиск пути с минимальным количеством промежуточных вершин (волновой алгоритм) На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины На выходе: path – маршрут между start и end (список вершин, входящих в маршрут) length – количество ребер маршрута	wave(self, start, end)
4	**Поиск минимального остовного дерева неориентированного графа по алгоритму Краскала** На входе: self – неориентированный граф На выходе: edges – список ребер, входящих в минимальное остовное дерево графа (список кортежей) weight – вес минимального остовного дерева	kruskal(self)
	Поиск минимального остовного дерева неориентированного графа по алгоритму Прима На входе: self – неориентированный граф На выходе: edges – список ребер, входящих в минимальное остовное дерево графа (список кортежей) weight – вес минимального остовного дерева	prima(self)
5	**Поиск минимального остовного дерева неориентированного графа по алгоритму Борувки** На входе: self – неориентированный граф На выходе: edges – список ребер, входящих в минимальное остовное дерево графа (список кортежей) weight – вес минимального остовного дерева	boruvka(self)
	Поиск пути с наибольшей пропускной способностью между двумя вершинами На входе: self – неориентированный граф start – номер стартовой вершины end – номер конечной вершины На выходе: path – маршрут между start и end с наибольшей пропускной способностью max_throughput – максимальная пропускная способность Примечание: для обхода графа используйте поиск в ширину (BFS)	throughput(self, start, end)
6	**Поиск простого центра и медианы неориентированного графа** На входе: self – неориентированный граф На выходе: centers – список центров графа medians – список медиан графа Примечание: для поиска кратчайших расстояний между вершинами графа используйте алгоритм Флойда-Уоршелла.	center(self) median(self)

	Поиск двойного простого центра неориентированного графа На входе: self – неориентированный граф На выходе: twin_centers – список пар центров графа twin_medians – список пар медиан графа Примечание: для поиска кратчайших расстояний между вершинами графа используйте алгоритм Флойда-Уоршелла.	double_center(self) double_median(self)
7	**Поиск внешнего, внутреннего и внешне-внутреннего центра ориентированного графа** На входе: self – неориентированный граф На выходе: outer_centers – список внешних центров графа inner_centers – список внутренних центров графа outer_inner_centers – список внешне-внутренних центров графа Примечание: для поиска кратчайших расстояний между вершинами графа используйте алгоритм Флойда-Уоршелла	center(self)
	Поиск сильных компонент ориентированного графа На входе: self – ориентированный граф На выходе: components – список сильных компонент графа Примечание: для обхода графа используйте поиск в глубину (DFS)	components(self)
8	**Поиск эйлерова цикла в неориентированном эйлеровом графе** На входе: self – неориентированный эйлеров граф На выходе: eulers_cycle – список вершин эйлерова цикла Примечание: для поиска эйлерова цикла используйте поиск в глубину (DFS)	eulers_cycle(self)
9	**Поиск точек сочленения графа (вершин, удаление которых приводит к распаду графа на 2 сильные компоненты)** На входе: self – неориентированный граф На выходе: cutpoints – список точек сочленения графа Примечание: для поиска точек сочленения графа используйте поиск в глубину (DFS)	cutpoints(self)
10	**Обход графа в глубину** На входе: self – неориентированный граф start – номер стартовой вершины На выходе: dfs_nodes – последовательность вершин при обходе в глубину dfs_edges – ребра в дереве обхода в глубину	dfs(self, start)
	Обход графа в ширину На входе: self – неориентированный граф start – номер стартовой вершины На выходе: dfs_nodes – последовательность вершин при обходе в ширину dfs_edges – ребра в дереве обхода в ширину	bfs(self, start)

Дополнительные темы для разработки:

1. Поиск максимального потока по алгоритму Форда-Фалкерсона.
2. Определение количества и расположения *p*-центров графа по заданному критическому расстоянию λ.
3. Поиск максимальной клики графа (полный подграф наибольшего размера).
4. Поиск максимального независимого множества вершин графа.

Задание №3 (можно выполнять вместо задания №2)

Напишите и протестируйте подкласс класса RandomGraph, позволяющий моделировать двудольный граф со взвешенными ребрами. Принять в двудольном графе приблизительно равные размеры долей. Подкласс RandomBiGraph должен содержать следующие методы, переопределяющие методы родительского класса:

class RandomGraph	
	class RandomBiGraph(RandomGraph) w_generation() – генерирует случайную матрицу весов W двудольного графа. hungary() - поиск максимального паросочетания в двудольном графе венгерским алгоритмом (решение задачи о назначениях)