Случайные процессы. Практическое задание 5

- Дедлайн **13 ноября 23:59** (13 дней на выполнение).
- Внимательно прочтите правила оформления. Задания, оформленные не по правилам, могут быть проигнорированы.
- В коде могут встречаться пропуски, которые обычно обозначаются так: <пояснение>
- Условие задания полностью серьезное. Шуток нет.

Для выполнения задания потребуются следующие библиотеки: bs4, urllib, networkx. Следующими командами можно их поставить (Ubuntu):

sudo pip3 install beautifulsoup4

sudo pip3 install urllib2

sudo pip3 install networkx

В случае возникновения проблем пишите на почту.

PageRank

История

(Взято с Википедии (https://ru.wikipedia.org/wiki/PageRank))

В 1996 году Сергей Брин и Ларри Пейдж, тогда ещё аспиранты Стэнфордского университета, начали работу над исследовательским проектом BackRub — поисковой системой по Интернету, использующей новую тогда идею о том, что веб-страница должна считаться тем «важнее», чем больше на неё ссылается других страниц, и чем более «важными», в свою очередь, являются эти страницы. Через некоторое время BackRub была переименована в Google. Первая статья с описанием применяющегося в ней метода ранжирования, названного PageRank, появилась в начале 1998 года, за ней следом вышла и статья с описанием архитектуры самой поисковой системы.

Их система значительно превосходила все существовавшие тогда поисковые системы, и Брин с Пейджем, осознав её потенциал, основали в сентябре 1998 года компанию Google Inc., для дальнейшего её развития как коммерческого продукта.

Описание

Введем понятие веб-графа. Ориентированный граф $G = (V, E)$ называется веб-графом, если

- $V = \{url_i\}_{i=1}^{n}$ --- некоторое подмножество страниц в интернете, каждой из которых соответствует адрес url_i.
- Множество E состоит из тех и только тех пар (url_i, url_j), для которых на странице с адресом url_i есть ссылка на url_j.

Рассмотрим следующую модель поведения пользователя. В начальный момент времени он выбирает некоторую страницу из V в соответствии с некоторым распределением $\Pi^{(0)}$. Затем, находясь на некоторой странице, он может либо перейти по какой-то ссылке, которая размещена на этой странице, либо выбрать случайную страницу из V и перейти на нее (damping factor). Считается, что если пользователь выбирает переход по ссылке, то он выбирает равновероятно любую ссылку с данной страницы и переходит по ней. Если же он выбирает переход не по ссылке, то он также выбирает равновероятно любую страницу из V и переходит на ее (в частности может остаться на той же странице). Будем считать, что переход не по ссылке пользователь выбирает с некоторой вероятностью $p \in (0, 1)$. Соответственно, переход по ссылке он выбирает с вероятностью $1 - p$. Если же со страницы нет ни одной ссылки, то будем считать, что пользователь всегда выбирает переход не по ссылке.

Описанная выше модель поведения пользователя называется моделью PageRank. Нетрудно понять, что этой модели соответствует некоторая марковская цепь. Опишите ее.

- Множество состояний: Страницы
- Начальное распределение: $\Pi^{(0)}$
- Переходные вероятности: $P = ||p_{ij}||$, где $p_{ij} = I(M > 0)(p/N + (1 - p)/M * I(i- > j)) + I(M = 0)1/N$, где N - количество страниц всего, M - количество ссылок со страницы i

Вычисление

Данная марковская цепь является эргодической. Почему?

Количество страниц, т.е. фазовое пространство, конечно и все элементы матрицы P >= p*1/N > 0, следовательно условия эргодической теоремы выполнены.

А это означает, что цепь имеет некоторое эргодическое распределение Π, которое является предельным и единственным стационарным. Данное распределение называется весом PageRank для нашего подмножества интернета.

Как вычислить это распределение Π для данного веб-графа? Обычно для этого используют степенной метод (power iteration), суть которого состоит в следующем. Выбирается некоторое начальное распределение $\Pi^{(0)}$. Далее производится несколько итераций по формуле $\Pi^{(k)} = \Pi^{(k-1)}P$, где P --- матрица переходных вероятностей цепи, до тех пор, пока $\left|\left|\Pi^{(k)} - \Pi^{(k-1)}\right|\right| > \varepsilon$. Распределение $\Pi^{(k)}$ считается приближением распредения Π.

Имеет ли смысл выполнять подобные итерации для разных начальных распределений $\Pi^{(0)}$ с точки зрения теории?

Нет, при любом начальном распределении предел будет тем же

А с точки зрения практического применения, не обязательно при этом доводя до сходимости?

Имеет, чтобы понять поведение цепи, в зависимости от начального распределения

Какая верхняя оценка на скорость сходимости?

Из доказательства теоремы - верхняя оценка : $(1 - min_{i,j}p_{ij})^n|max_i p_{ij} - min_i p_{ij}|$

За ответы можно получить **1 балл**.

Часть 1

За выполнение этой части можно получить **2 балла** за автоматическую проверку и **2 балла** за все остальное.

In [1]:
```
import numpy as np
from scipy.stats import bernoulli
import networkx
from bs4 import BeautifulSoup
from urllib.request import urlopen
from urllib.parse import urlparse, urlunparse
from time import sleep
import html5lib
from itertools import product
import matplotlib.pyplot as plt

%matplotlib inline
```

Реализуйте вычисление весов PageRank power-методом.

Реализовать может быть удобнее с помощью функции np.nan_to_num, которая в данном numpy.array заменит все вхождения nan на ноль. Это позволяет удобно производить поэлементное деление одного вектора на другой в случае, если во втором векторе есть нули.

In [2]:

```
def create_page_rank_markov_chain(links, damping_factor=0.15):
    ''' По веб-графу со списком ребер links строит матрицу
    переходных вероятностей соответствующей марковской цепи.

        links --- список (list) пар вершин (tuple),
                может быть передан в виде numpy.array, shape=(|E|, 2);
        damping_factor --- вероятность перехода не по ссылке (float);

        Возвращает prob_matrix --- numpy.matrix, shape=(|V|, |V|).
    '''

    links = np.array(links)
    N = links.max() + 1  # Число веб-страниц

    prob_matrix = np.ones((N, N))
    # без переходов по ссылкам
    prob_matrix *= 1/N

    #каждый i-ый элемент - список исходящих ребер из i
    edges = [[] for i in np.arange(N)]
    for e in links:
        edges[e[0]].append(e[1])

    # добавим переходы
    for i in np.arange(len(edges)):
        M = len(edges[i])
        if(M>0):
            prob_matrix[i, :] *= damping_factor
            for j in edges[i]:
                prob_matrix[i, j] += (1-damping_factor)/M

    return np.matrix(prob_matrix)


def page_rank(links, start_distribution, damping_factor=0.15,
              tolerance=10 ** (-7), return_trace=False):
    ''' Вычисляет веса PageRank для веб-графа со списком ребер links
    степенным методом, начиная с начального распределения start_distribution,
    доводя до сходимости с точностью tolerance.

        links --- список (list) пар вершин (tuple),
                может быть передан в виде numpy.array, shape=(|E|, 2);
        start_distribution --- вектор размерности |V| в формате numpy.array;
        damping_factor --- вероятность перехода не по ссылке (float);
        tolerance --- точность вычисления предельного распределения;
        return_trace --- если указана, то возвращает список распределений во
                            все моменты времени до сходимости

        Возвращает:
        1). если return_trace == False, то возвращает distribution ---
        приближение предельного распределения цепи,
        которое соответствует весам PageRank.
        Имеет тип numpy.array размерности |V|.
        2). если return_trace == True, то возвращает также trace ---
        список распределений во все моменты времени до сходимости.
        Имеет тип numpy.array размерности
        (количество итераций) на |V|.
    '''

    prob_matrix = create_page_rank_markov_chain(links,
                                                damping_factor=damping_fact
```

```
or)
    distribution = np.matrix(start_distribution)
    #в trace  логично добавлять и начальное, и конечное распределение, но что имеется
 в виду в задании не ясно
    trace = []
    while(True):
        trace.append(distribution)
        distribution = np.dot(distribution, prob_matrix)
        if (np.linalg.norm(distribution - trace[-1]) < tolerance):
            break

    trace.append(distribution)

    if return_trace:
        return np.array(distribution).ravel(), np.array(trace)
    else:
        return np.array(distribution).ravel()
```

Автоматическая проверка (2 балла)

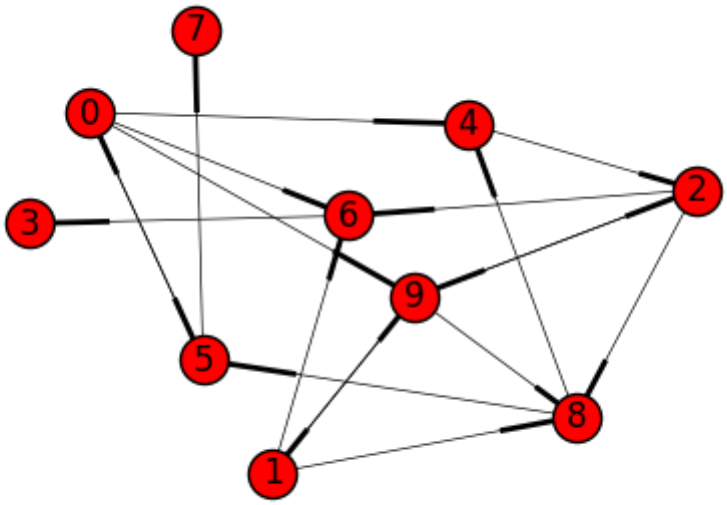
Реализацию функций create_page_rank_markov_chain и page_rank скопируйте в файл с названием v[номер варианта].py и вышлите на почту. Будет проверяться только корректность выдаваемых значений. Проверки на время работы не будет.

Давайте посмотрим, как оно работает. Напишите для начала функцию для генерации случайного ориентированного графа $G(n, p)$. Случайный граф генерируется следующий образом. Берется множество $\{0, \ldots, n-1\}$, которое есть множество вершин этого графа. Ребро (i, j) (пара упорядочена, возможно повторение) добавляется в граф независимо от других ребер с вероятностью p.

In [3]:
```
def random_graph(n, p):
    return [(j, i)for i in np.arange(n) for j in np.arange(n) if (bernoulli.
(p) == 1)]
```

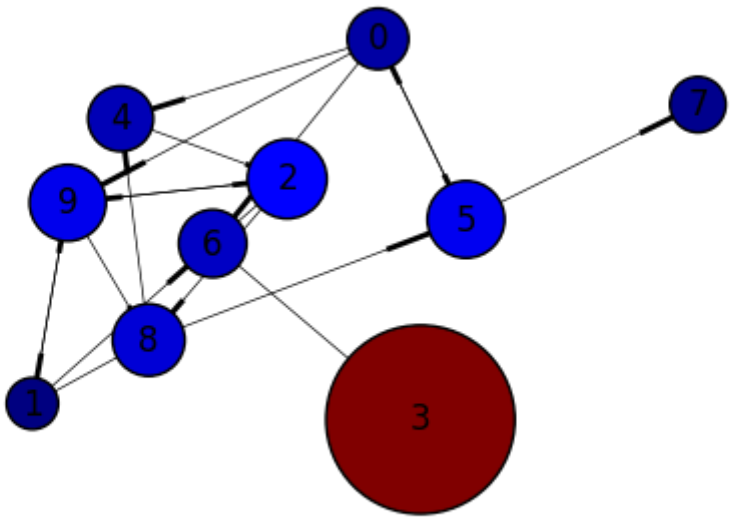
Теперь сгенерируем случайный граф и нарисуем его.

In [50]:
```
N, p = 10, 0.2
edges = random_graph(N, p)

G = networkx.DiGraph()
G.add_edges_from(edges)
plt.axis('off')
networkx.draw_networkx(G, width=0.5)
```



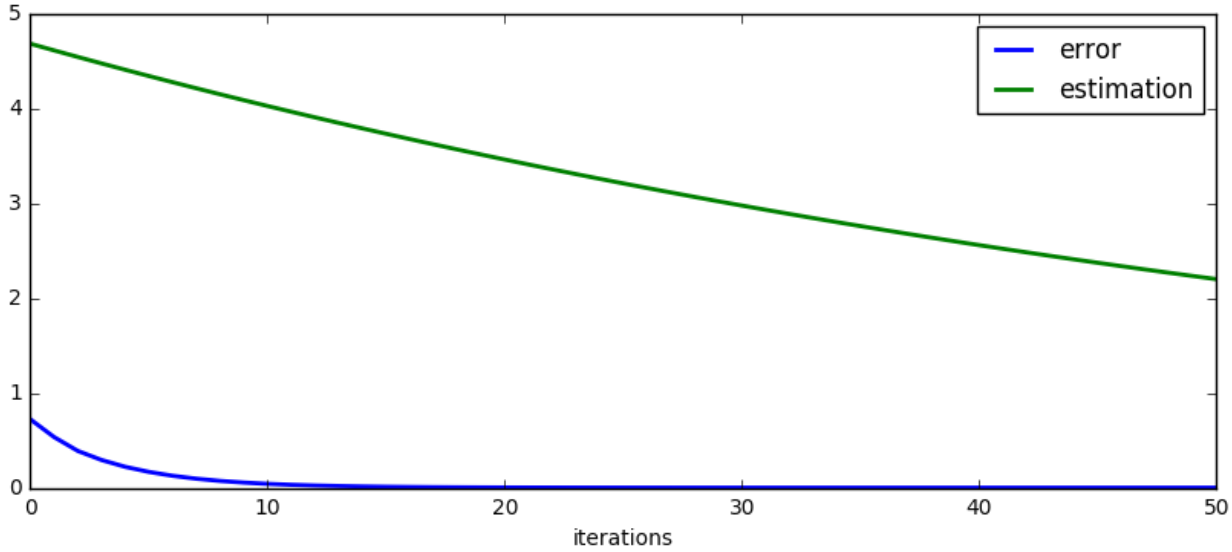
Посчитаем его PageRank и изобразим так, чтобы размер вершины был пропорционален ее весу.

In [51]:
```
start_distribution = np.ones((1, N)) / N
pr_distribution= page_rank(edges, start_distribution)

size_const = 10 ** 4
plt.axis('off')
networkx.draw_networkx(G, width=0.5, node_size=size_const *
pr_distribution,
                       node_color=pr_distribution)
```



Как мы уже отмечали выше, эргодическая теорема дает верхнюю оценку на скорость сходимости. Давайте посмотрим, насколько она является точной. Для этого при вычислении PageRank нужно установить флаг return_trace.

In [54]:
```
pr_distribution, pr_trace = page_rank(edges, start_distribution,
                                      return_trace=True)
errors = np.abs(pr_trace - pr_trace[-1]).sum(axis=(1, 2))

P = create_page_rank_markov_chain(edges)

eps = np.min(P)
differ = np.linalg.norm(np.max(P, axis=0) - np.min(P, axis=1))
plt.figure(figsize=(10, 4))
x = np.arange(len(errors))
plt.plot(x, errors, lw=2, label='error')
plt.plot(x, (1-eps) ** x * differ, lw=2, label='estimation')
plt.legend()
plt.xlabel('iterations')
plt.show()
```



Эргодическая теорема дает оценку на скорость сходимости с большим запасом, на практике сходимость более быстрая

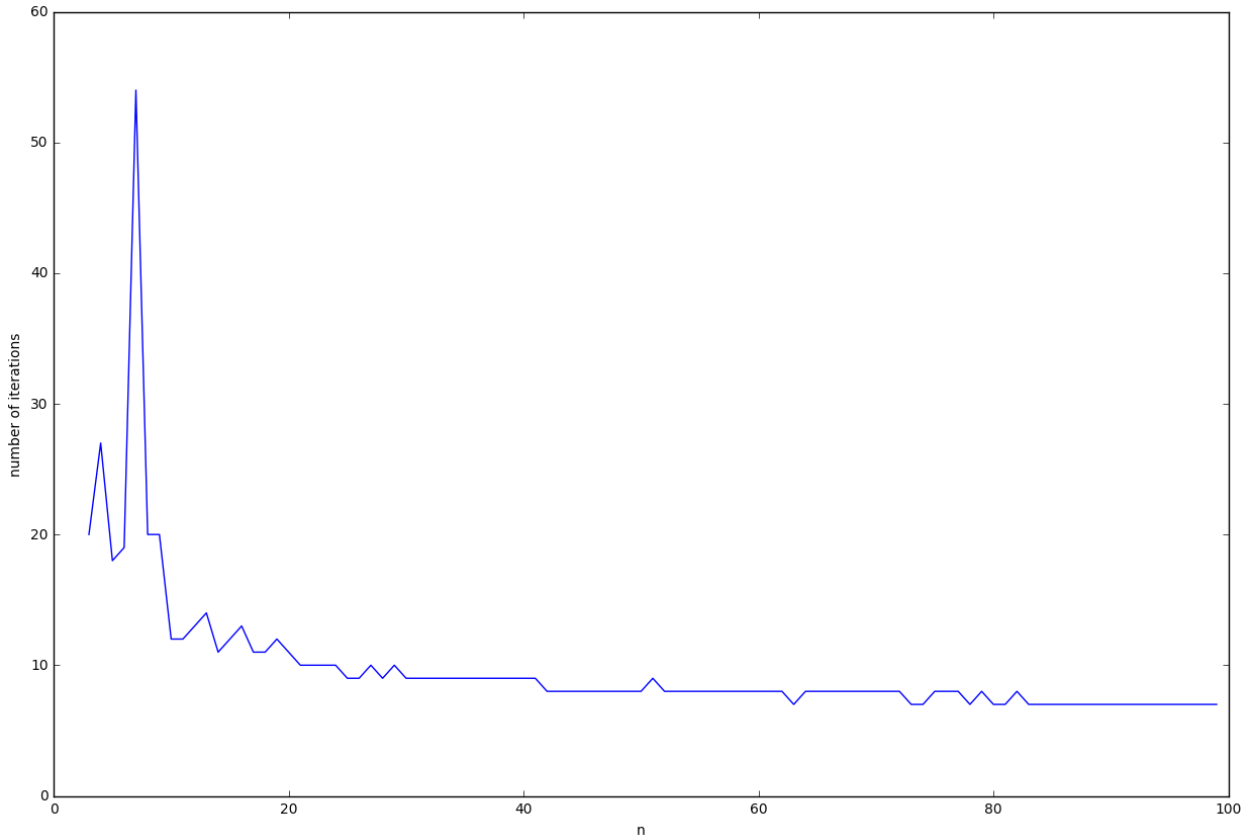
Проведите небольшое исследование. В ходе исследования выясните, как скорость сходимости (количество итераций до сходимости) зависит от n и p, а так же начального распределения. Вычислите также веса PageRank для некоторых неслучайных графов. В каждом случае стройте графики. От чего зависит вес вершины?

Зависимость от n

фиксируем p и начальное распределение

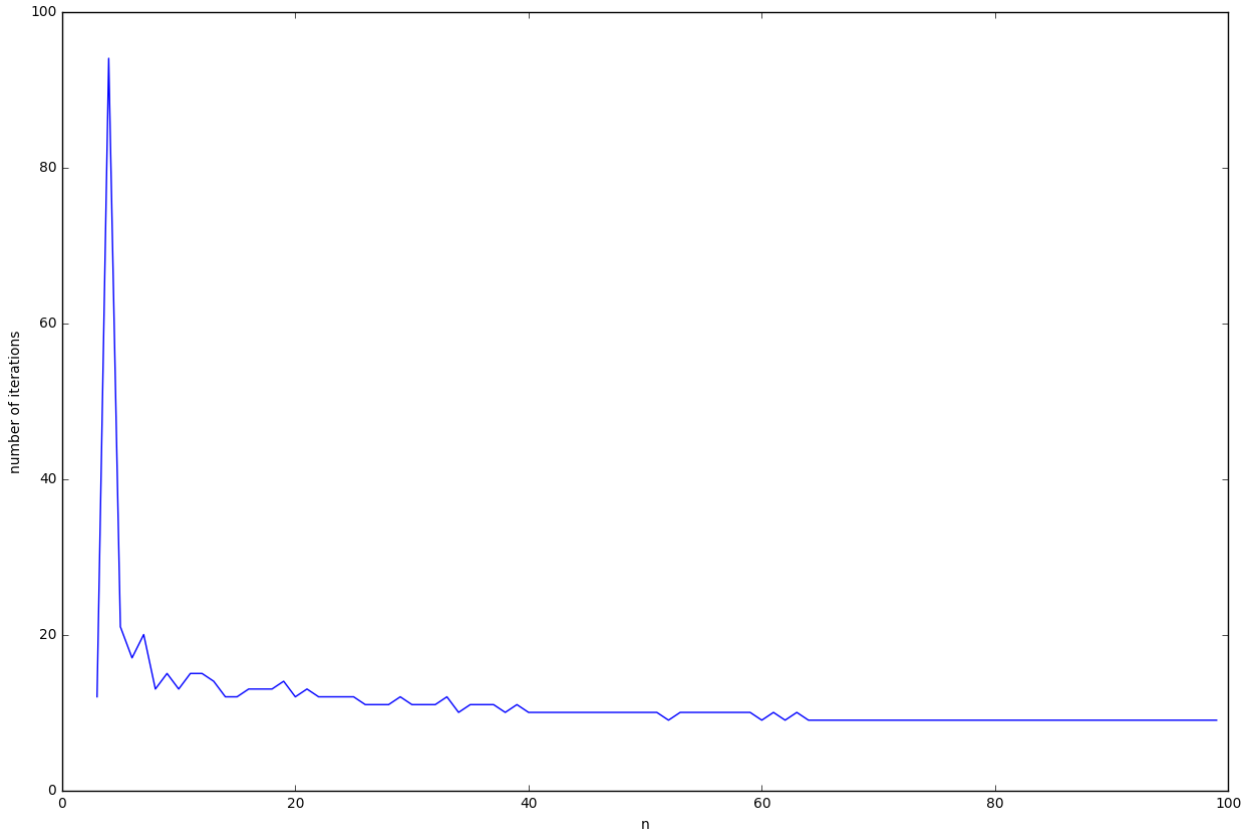
In [69]:
```
grid = np.arange(3, 100)
iters = np.zeros_like(grid)
for i in grid:
    edges = random_graph(i, 0.5)
    start_distribution = np.ones((1, i)) / i
    distr, trace = page_rank(edges, start_distribution, return_trace=True)
    iters[i-3] = trace.shape[0]
```

In [70]:
```
plt.figure(figsize=(15,10))
plt.plot(grid, iters)
plt.xlabel('n')
plt.ylabel('number of iterations')
plt.show()
```



На графике видно, что при фиксированных p и начальном распределении при росте n количество итераций уменьшается, это связано с тем, что с ростом n граф становится более симметричным в данной модели случайного графа, поэтому симметричное распределение все лучше приближает его. Рассмотрим другое начальное распределение

In [79]:
```
grid = np.arange(3, 100)
iters = np.zeros_like(grid)
for i in grid:
    edges = random_graph(i, 0.5)
    start_distribution = np.zeros((1, i))
    start_distribution[0][0] += 1/3
    start_distribution[0][1] += 1/3
    start_distribution[0][2] += 1/3
    distr, trace = page_rank(edges, start_distribution, return_trace=True)
    iters[i-3] = trace.shape[0]
plt.figure(figsize=(15,10))
plt.plot(grid, iters)
plt.xlabel('n')
plt.ylabel('number of iterations')
plt.show()
```



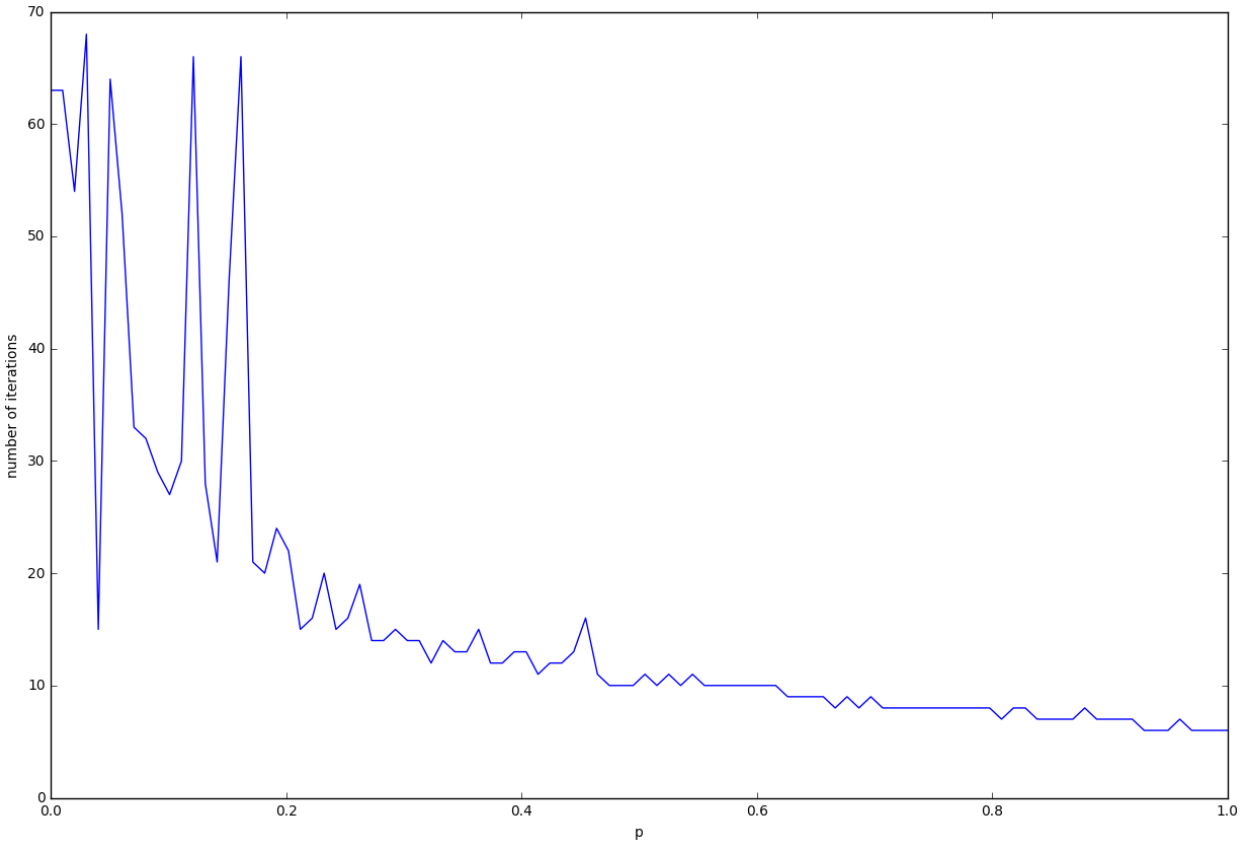
В данном случае функция стремится к константе, что понятно, тк рспределение стремится к равномерному, чтобы прийти к нему нужно будет сделать определенное число шагов

Зависимость от p

фиксируем n и начальное распределение

In [76]:
```
grid = np.linspace(0, 1, 100)
iters = np.zeros_like(grid)
j = 0
for i in grid:
    edges = random_graph(20, i)
    edges.append((19,19))
    start_distribution = np.ones((1, 20)) / 20
    if(len(edges)==0):
        iters[j] = 0
    else:
        distr, trace = page_rank(edges, start_distribution, return_trace=True)
        iters[j] = trace.shape[0]
    j += 1
```

In [77]:
```
plt.figure(figsize=(15,10))
plt.plot(grid, iters)
plt.xlabel('p')
plt.ylabel('number of iterations')
plt.show()
```

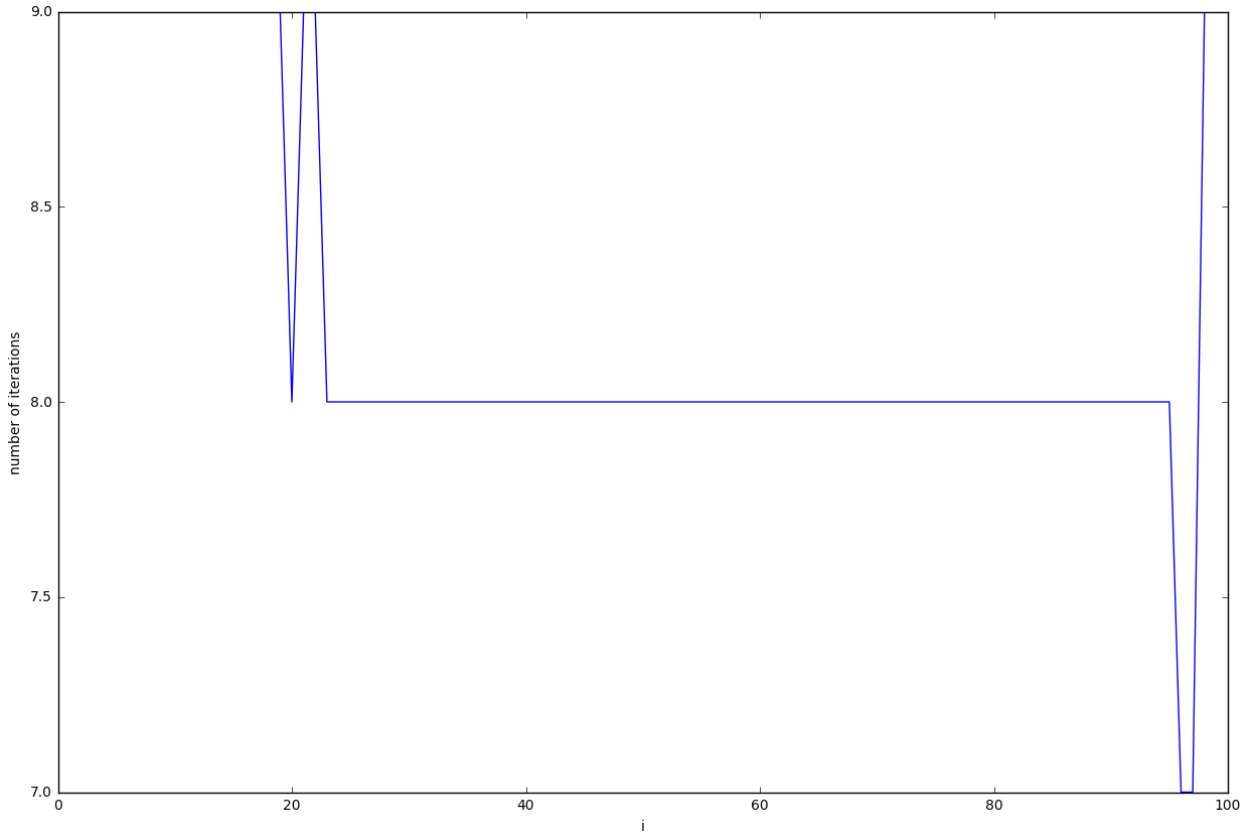


На графике видно, что с ростом p количество итераций уменьшается, это также связано с тем, что граф становится более симметричным с ростом p

Зависимость от начального распределения

фиксируем n и p, рассмотрим различные начальные распределения в зависимости от i - числа объектов, из которых в начале будем делать выбор разновероятно

In [83]:
```
grid = np.arange(1, 100)
edges = random_graph(100, 0.5)
iters = np.zeros_like(grid)
for i in grid:
    start_distribution = np.zeros((1, 100))
    for j in np.arange(i):
        start_distribution[0][j] += 1/i
    distr, trace = page_rank(edges, start_distribution, return_trace=True)
    iters[i-3] = trace.shape[0]

plt.figure(figsize=(15,10))
plt.plot(grid, iters)
plt.xlabel('i')
plt.ylabel('number of iterations')
plt.show()
```



зависимость не просматривается, число итераций почти одинаково, следовательно, можно брать любое приближение

Часть 2

За выполнение этой части можно получить **5 баллов** и больше. В этой части вам предстоит построить реальный веб-граф и посчитать его PageRank. Ниже определены вспомогательные функции.

In [9]:
```
def load_links(url, sleep_time=1, attempts=5, timeout=20):
    ''' Загружает страницу по ссылке url и выдает список ссылок,
    на которые ссылается данная страница.
        url --- string, адрес страницы в интернете;
        sleep_time --- задержка перед загрузкой страницы;
        timeout --- время ожидания загрузки страницы;
        attempts --- число попыток загрузки страницы.
        Попытка считается неудачной, если выбрасывается исключение.

        В случае, если за attempts попыток не удалось загрузить страницу,
        то последнее исключение пробрасывается дальше.
    '''

    sleep(sleep_time)
    parsed_url = urlparse(url)
    links = []

    # Попытки загрузить страницу
    for i in range(attempts):
        try:
            # Ловить исключения только из urlopen может быть недостаточно.
            # Он может выдавать какой-то бред вместо исключения,
            # из-за которого исключение сгенерирует BeautifulSoup
            soup = BeautifulSoup(urlopen(url, timeout=timeout), 'lxml')
            break

        except Exception as e:
            print(e)
            if i == attempts - 1:
                raise e

    for tag_a in soup('a'):  # Посмотр всех ссылочных тегов
        if 'href' in tag_a.attrs:
            link = list(urlparse(tag_a['href']))

            # Если ссылка является относительной,
            # то ее нужно перевести в абсолютную
            if link[0] == '': link[0] = parsed_url.scheme
            if link[1] == '': link[1] = parsed_url.netloc

            links.append(urlunparse(link))

    return links


def get_site(url):
    ''' По ссылке url возвращает адрес сайта. '''

    return urlparse(url).netloc
```

Код ниже загружает N веб-страниц, начиная с некоторой стартовой страницы и переходя по ссылкам. Загрузка происходит методом обхода в ширину. Все собранные урлы страниц хранятся в urls. В links хранится список ссылок с одной страницы на другую. Особенность кода такова, что в urls хранятся все встреченные урлы, которых может быть сильно больше N. Аналогично, в links ребра могут ссылаться на страницы с номером больше N. Однако, все ребра из links начинаются только в первых N страницах. Таким образом, для построения веб-графа нужно удалить все, что связано с вершинами, которые не входят в первые N.

Это очень примерный шаблон, к тому же не оптимальный. Можете вообще его не использовать и написать свое.

In [36]:
```
urls = ['http://ru.discrete-mathematics.org/']
site = get_site(urls[0])
links = []

i=0
#код модифицирован для загрузки всего сайта
while(i < len(urls)):
    try:
        # Загружаем страницу по урлу и извлекаем из него все ссылки
        # Не выставляйте sleep_time слишком маленьким,
        # а то еще забанят где-нибудь
        links_from_url = load_links(urls[i], sleep_time=0.5)
        # Если мы хотим переходить по ссылкам только определенного сайта
        links_from_url = list(filter(lambda x: get_site(x) == site,
                                     links_from_url))

        # Добавляем соответствующие вершины и ребра в веб-граф
        for j in range(len(links_from_url)):
            # Такая ссылка уже есть
            if links_from_url[j] in urls:
                links.append((i, urls.index(links_from_url[j])))

            # Новая ссылка
            else:
                links.append((i, len(urls)))
                urls.append(links_from_url[j])

    except:
        pass  # Не загрузилась с 5 попытки, ну и ладно
    i+=1
```

```
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
'ascii' codec can't encode characters in position 32-36: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-36: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-36: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-36: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-36: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-37: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-37: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-37: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-37: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-37: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-38: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-38: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-38: ordinal not in ra
nge(128)
```

```
'ascii' codec can't encode characters in position 32-38: ordinal not in ra
nge(128)
'ascii' codec can't encode characters in position 32-38: ordinal not in ra
nge(128)
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 403: Forbidden
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
<urlopen error unknown url type: mailto>
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
```

HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found
HTTP Error 404: Not Found

Теперь выберите какой-нибудь сайт с небольшим количеством страниц (не более 1000). Таким сайтом может быть, например, сайт кафедры (http://ru.discrete-mathematics.org>кафедры) Дискретной математики (аккуратнее, если забанят, то лишитесь доступа к учебным материалам), Школы (http://yandexdataschool.ru>Школы) анализа данных, сайт магазина, больницы. Однако, советуем не выбирать сайты типа kremlin.ru, мало ли что.

Постройте полный веб-граф для этого сайта и визуализируйте его. При отрисовке выставляйте width не более 0.1, иначе получится ужасно некрасиво.

Посчитайте PageRank для этого веб-графа. Визуализируйте данный веб-граф, сделав размер вершин пропорционально весу PageRank (см. пример в части 1). Постройте гистограмму весов. Что можно сказать про скорость сходимости?

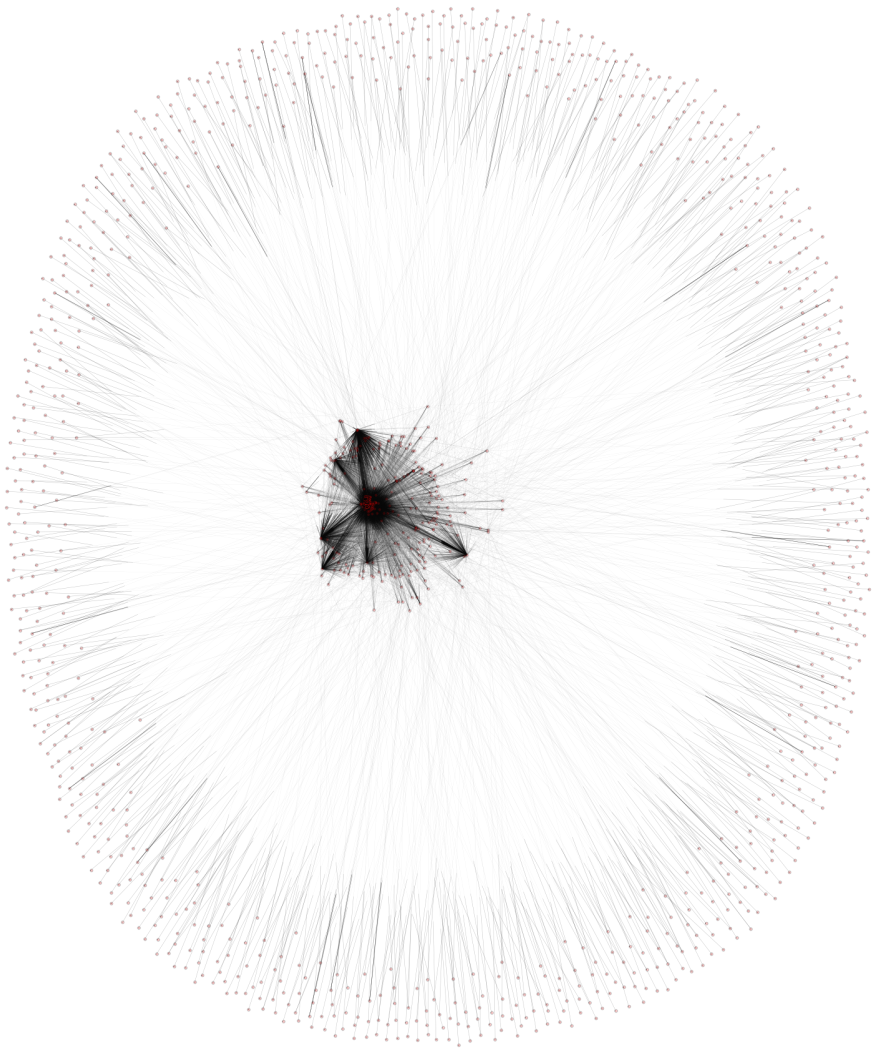
Выделите небольшое количество (15-20) страниц с наибольшим весом и изобразите граф, индуцированный на этом множестве вершин. Что это за страницы? Почему именно они имеют большой вес?

Как меняется вес PageRank для страниц в зависимости от начального приближения в случае, если не доводить итерационный процесс вычисления до сходимости? Какие выводы о поведении пользователя отсюда можно сделать?

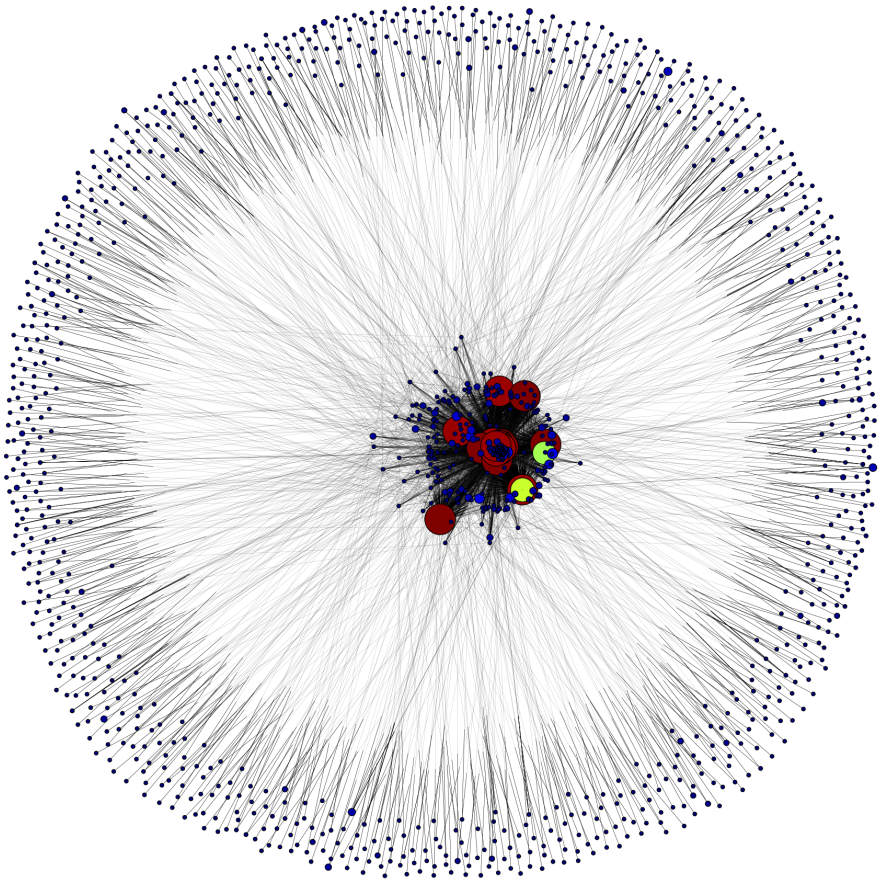
Для получения дополнительных баллов проведите аналогичные исследования для больших сайтов. Так же вы можете провести исследования, не ограничиваясь загрузкой только одного сайта.

In [101]:
```
G = networkx.DiGraph()
G.add_edges_from(links)
plt.figure(figsize=(25, 30))
plt.axis('off')
plt.title("Web graph for ru.discrete-mathematics.org")
networkx.draw_networkx(G, width=0.1, alpha=0.2, font_size=0, node_size=10)
```

Web graph for ru.discrete-mathematics.org

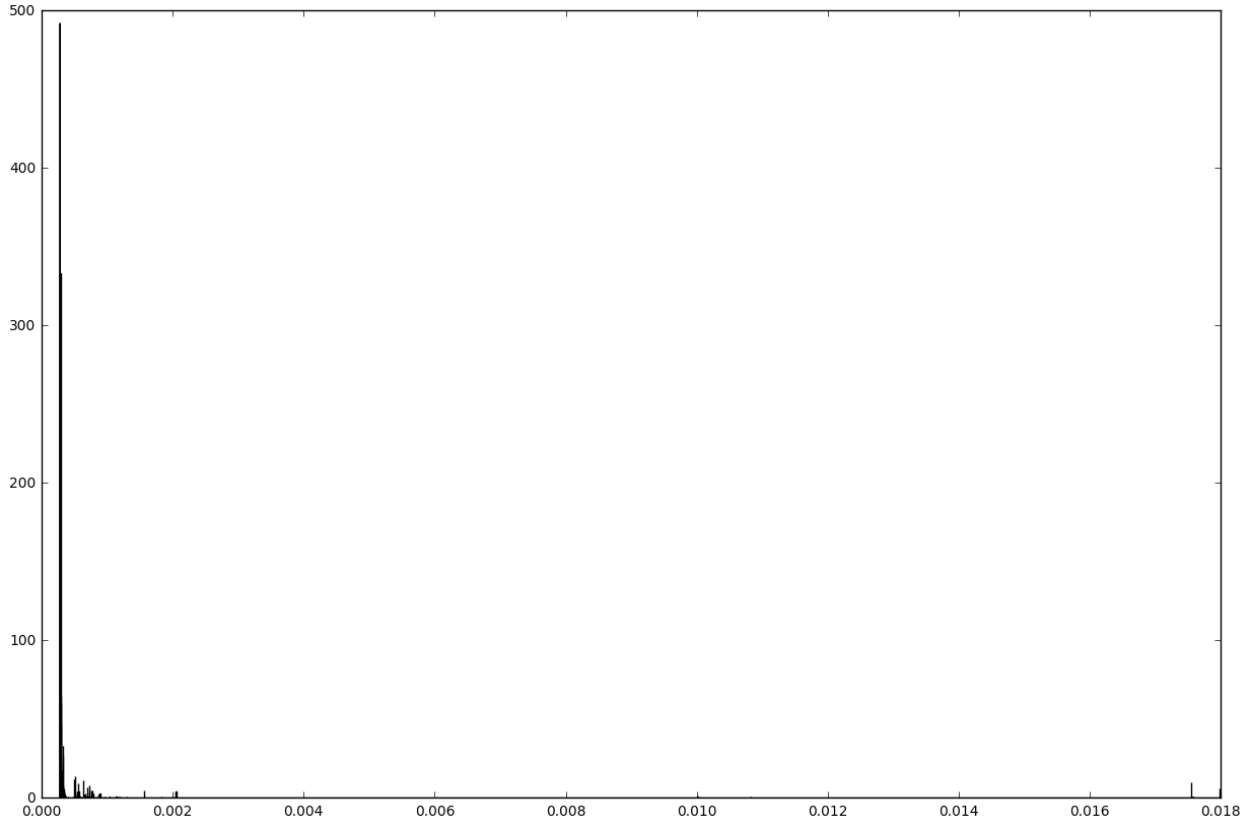


In [102]:
```
N = len(urls)
start_distribution = np.ones((1, N)) / N
pr_distribution = page_rank(links, start_distribution)

size_const = 10 ** 5
plt.figure(figsize=(30, 30))
plt.axis('off')
networkx.draw_networkx(G, width=0.1, node_size=size_const *
pr_distribution,
                       node_color=pr_distribution, font_size=0)
```

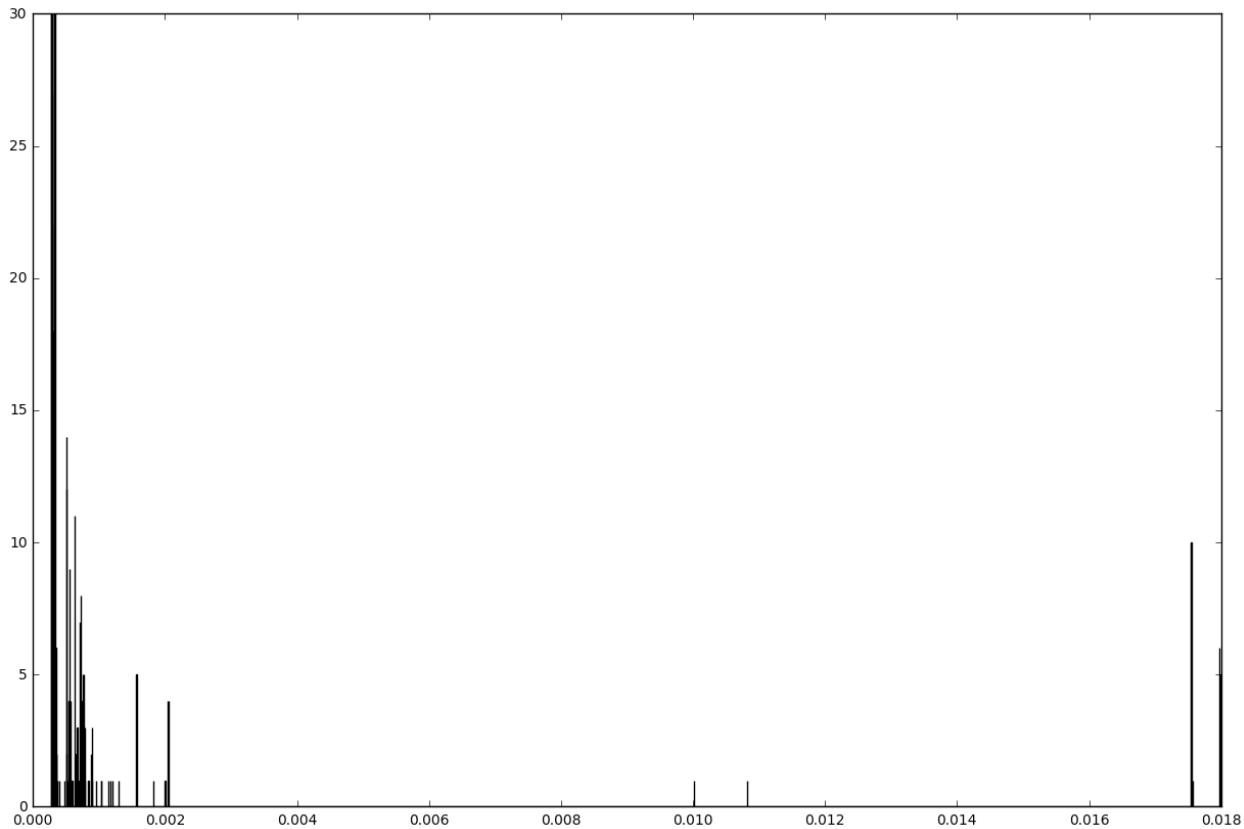


Гистограмма распределения весов

In [103]:
```
plt.figure(figsize=(15, 10))
plt.hist(pr_distribution, bins='auto')
plt.show()
```



In [104]:
```
plt.figure(figsize=(15, 10))
plt.hist(pr_distribution, bins='auto')
plt.ylim((0, 30))
plt.show()
```



На гистограмме видно, что подавляющее большинство объектов имеют очень маленький одинаковый вес, поэтому изначальное равномерное приближение очень близко к правде, и алгоритм приближения сходится очень быстро

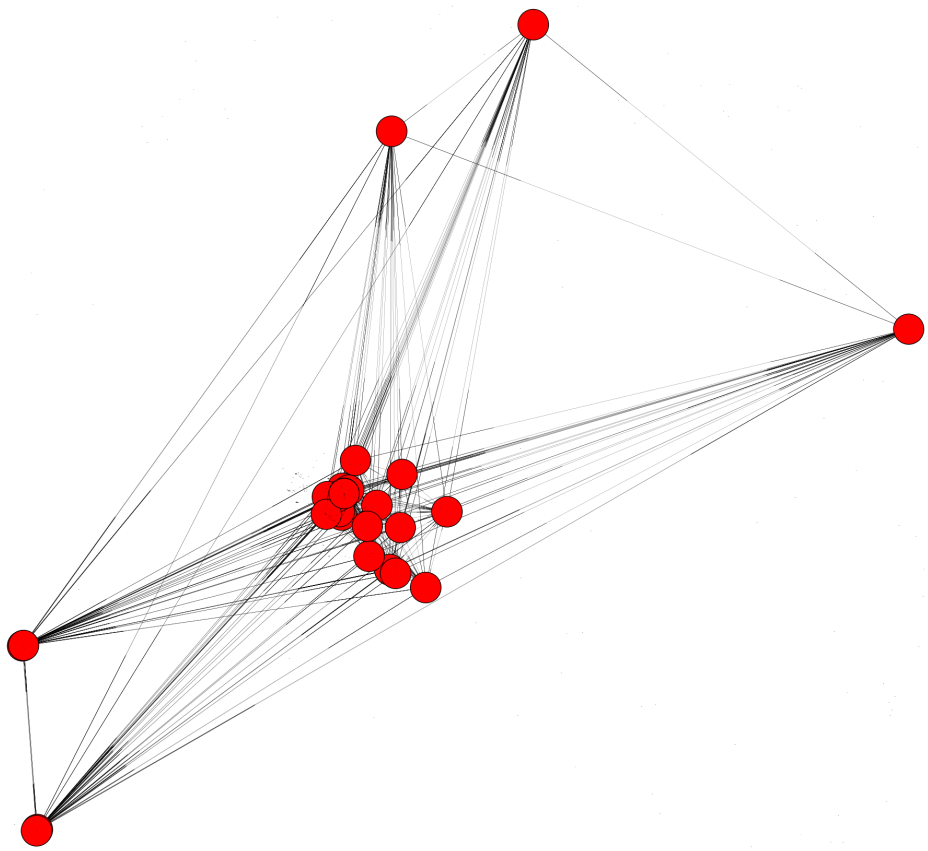
Найдем страницы с наибольшими весами

In [105]:
```
maxweigth = [i for i in np.arange(len(pr_distribution)) if pr_distribution[i
 > 0.016]
```

In [107]:
```
len(maxweigth)
```

Out[107]: 29

In [113]:
```
size_const = 10 ** 5
edgeelist = [k for k in links if((k[0] in maxweigth) and (k[1] in
maxweigth))]
plt.figure(figsize=(30, 30))
plt.axis('off')
networkx.draw_networkx(G, width=0.1, node_size=size_const *
pr_distribution, font_size=0, nodelist=maxweigth, edgelist = edgeelist)
```



In [128]:
```
print([urls[i] for i in maxweigth])
```

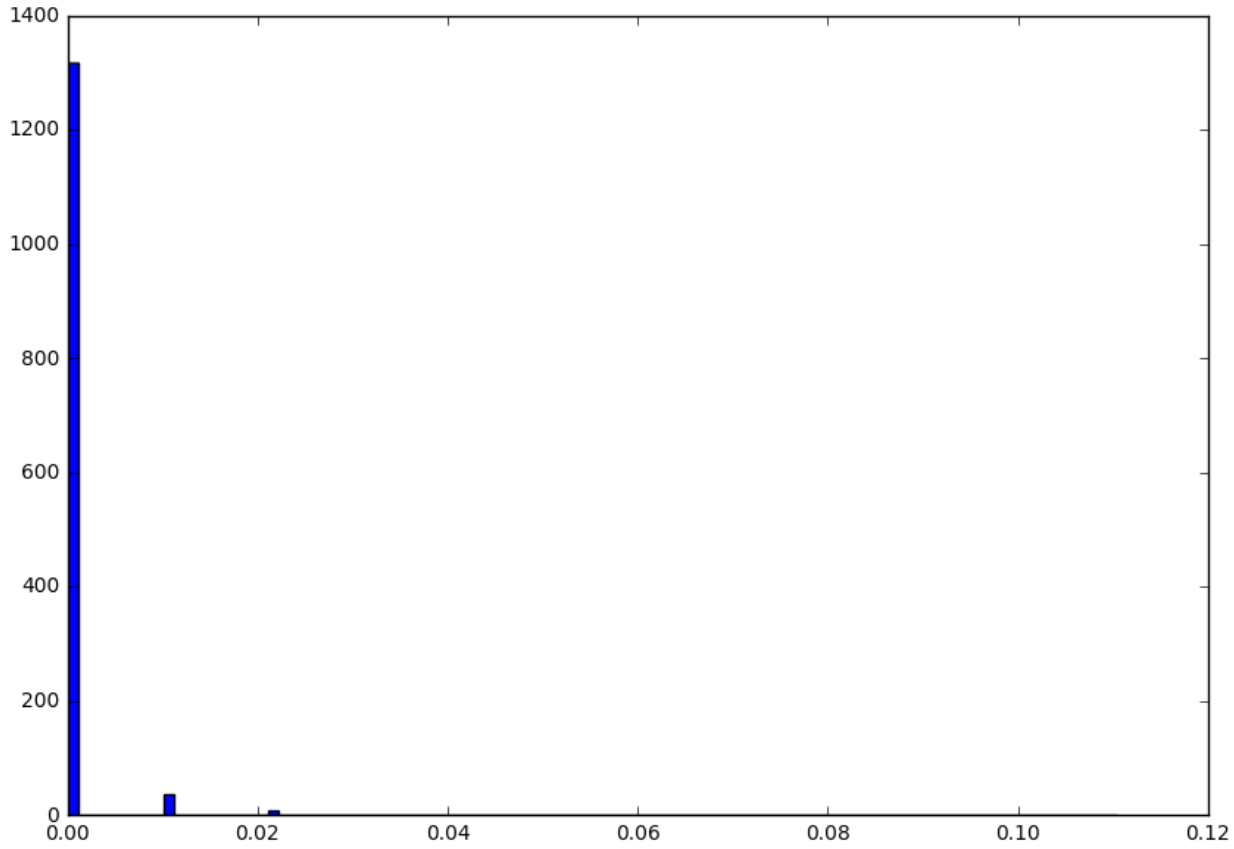
```
['http://ru.discrete-mathematics.org/', 'http://ru.discrete-mathematics.or
g#content', 'http://ru.discrete-mathematics.org/?page_id=41', 'http://ru.d
iscrete-mathematics.org/?page_id=27', 'http://ru.discrete-mathematics.or
g/?page_id=1414', 'http://ru.discrete-mathematics.org/?page_id=599', 'htt
p://ru.discrete-mathematics.org/?page_id=9', 'http://ru.discrete-mathemati
cs.org/?page_id=2', 'http://ru.discrete-mathematics.org/?page_id=624', 'ht
tp://ru.discrete-mathematics.org/?page_id=5', 'http://ru.discrete-mathemat
ics.org/?page_id=252', 'http://ru.discrete-mathematics.org/?page_id=394',
 'http://ru.discrete-mathematics.org/?page_id=449', 'http://ru.discrete-ma
thematics.org/?page_id=699', 'http://ru.discrete-mathematics.org/?page_id=
625', 'http://ru.discrete-mathematics.org/?page_id=626', 'http://ru.discre
te-mathematics.org/?page_id=3346', 'http://ru.discrete-mathematics.org/?pa
ge_id=326', 'http://ru.discrete-mathematics.org/?page_id=2909', 'http://r
u.discrete-mathematics.org/?page_id=320', 'http://ru.discrete-mathematics.
org/?page_id=3270', 'http://ru.discrete-mathematics.org/?page_id=3194', 'h
ttp://ru.discrete-mathematics.org/?page_id=3191', 'http://ru.discrete-math
ematics.org/?page_id=3196', 'http://ru.discrete-mathematics.org/?page_id=4
54', 'http://ru.discrete-mathematics.org/?page_id=300', 'http://ru.discret
e-mathematics.org/?page_id=302', 'http://ru.discrete-mathematics.org/?page
_id=305', 'http://ru.discrete-mathematics.org/?page_id=2332']
```

На эти страницы много ссылок, поэтому они имеют больший вес - например, главная страница, страница англоязычтой магистратуры, межкаф семинара и тд

Рассмотрим другие начальные приближения - для реалистичности буду рассматривать в качестве начальных страниц наиболее популярные

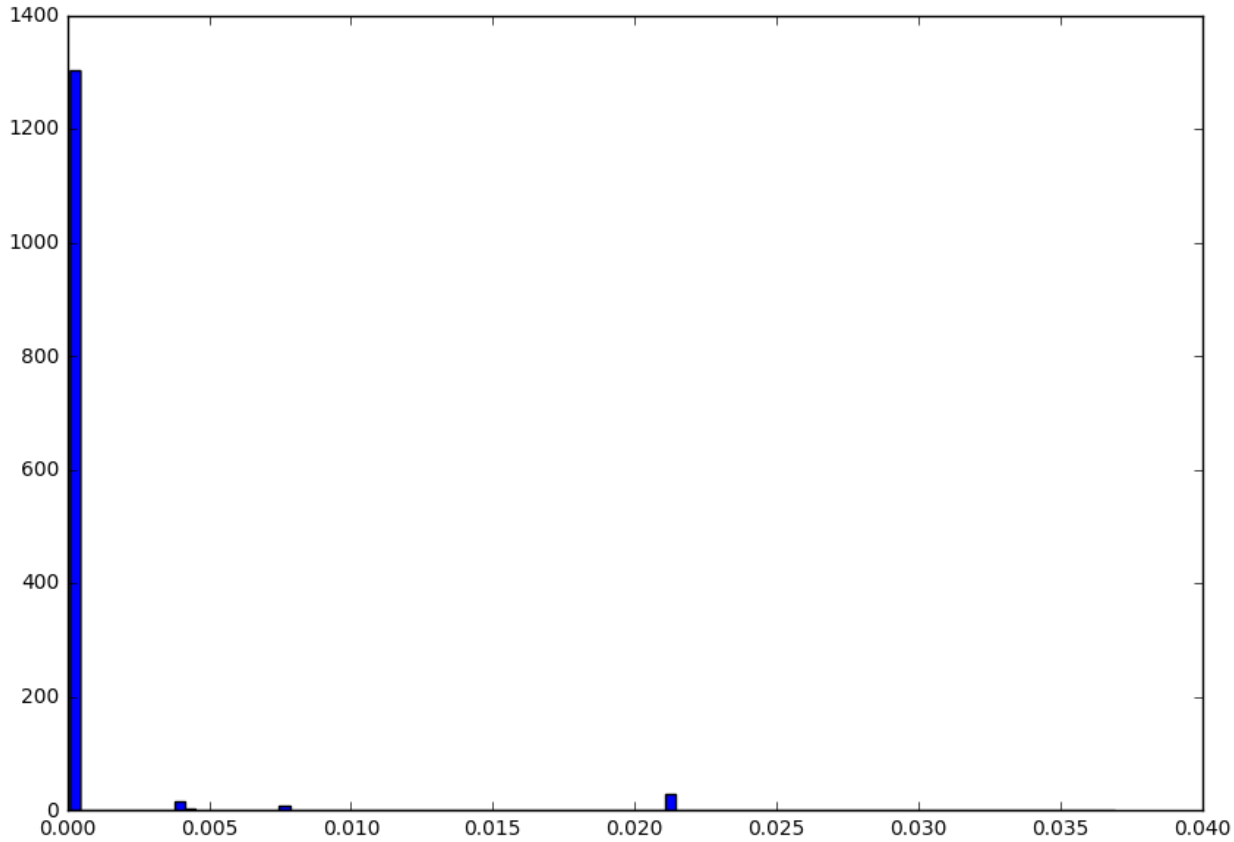
С вероятностью 1 выберем главную страницу (самый популярный сценарий)

In [123]:
```
start_distribution = np.zeros((1, N))
start_distribution[0][0] = 1
pr_distribution = page_rank(links, start_distribution, tolerance=2)
plt.figure(figsize=(10, 7))
plt.hist(pr_distribution, bins=100)
plt.show()
```



получается, на большинство страниц, мы так и не зайдем. теперь поменяем распределение: дадим большой вез главной странице и веса поменьше страницам 2,3,4,5 курсов

In [130]:
```
start_distribution = np.zeros((1, N))
start_distribution[0][0] = 1./3
start_distribution[0][10] = 1./6
start_distribution[0][11] = 1./6
start_distribution[0][12] = 1./6
start_distribution[0][13] = 1./6
pr_distribution = page_rank(links, start_distribution, tolerance=2)
plt.figure(figsize=(10, 7))
plt.hist(pr_distribution, bins=100)
plt.show()
```



Из данной гистограммы можно сделать вывод, что, как и в прошлый раз пользователь точно не попадет на большую часть страниц

Получается, что, если не доводить до сходимости, то большинство весов не изменяются