HW3

January 29, 2022

1 Homework 3

1.0.1 Released: 01/18/22

1.0.2 Due Date: 01/31/22

1.0.3 Name: Joanne Qiu

1.0.4 Collaborators: N/A

This homework focuses on topics related to classes, inheritance, exceptions, and iterators.

I encourage collaborating with your peers, but the final text, code, and comments in this homework assignment should still be written by you.

Submission instructions: - Convert this notebook into a pdf file and submit it on GradeScope under "HW3 - PDF". - Compress files `PageRank.py`, and `utils.py` into one `HW3.zip`, and submit it to Gradescope under "HW3 - Autograder". Make sure the python files are at the top of the directory when you zip them, and don't change their name.

1.0.5 PageRank

What is the most important website on the internet? Who is the "key player" on a sports team? Which countries are the most central players in the world economy? There is no one correct answer to any of these questions, but there is a most profitable one. PageRank is an algorithm for ranking individual elements of complex systems, invited by Sergey Brin and Larry Page. It was the first and most famous algorithm used by the Google Search engine, and it is fair to say that the internet as we know it today would not exist without PageRank.

In this assignment, we will implement PageRank. There are many good ways to implement this algorithm, but in this assignment we will use our newfound skills with object-oriented programming and iterators.

1.0.6 How it works

For the purposes of this example, let's assume that we are talking about webpages. PageRank works by allowing a "random surfer" to move around webpages by following links. Each time the surfer lands on a page, it then looks for all the links on that page. It then picks one at random and follows it, thereby arriving at the next page, where the process repeats. Because the surfer moves between linked pages, PageRank expresses an intuitive idea: **important pages are linked to other important pages.** This diagram from Wikipedia gives a nice illustration. Note that more important webpages (higher PageRank) tend to be connected to other important webpages.

A schematic for PageRank.

(There's a small issue related to webpages that don't have any links, which we'll address later on in the assignment. This is the "damping factor" in the diagram, which comes into play when we implement "teleportation" below.)

1.0.7 Data

This data set comes from the hit Broadway musical "Hamilton."

The Hamilton data set

The good folks at The Hamilton Project analyzed the script for us, obtaining data on **who talks about whom** in each of the show's songs. When character A mentions character B, we'll think of this as a *link* from A to B, suggesting that B might be important.

Listening to the soundtrack while working is strongly recommended.

[1]:
```
# import functions and classes from other files.
from utils import retrieve_data, read_data, describe, data_to_dictionary
from PageRank import PageRankDiGraph, PageRankIterator
```

1.1 Problem 1

1.1.1 Part (a): Call `retrieve_data` and `read_data` with proper arguments

In `utils.py`, take a look at the two functions that are already defined. The first one `retrieve_data` retrieves the data from the internet and saves it to your local computer, while the second `read_data` reads in the data from the local copy, producing a list of tuples.

The Hamilton dataset lives at the following URL:

`https://philchodrow.github.io/PIC16A/homework/HW3-hamilton-data.csv`

Each row corresponds to a "link" between objects, and the pairs have format `mentioner, mentioned`.

The cell below

- sets the variable `url`,
- calls `retrieve_data` and `read_data` with proper arguments
- saves the return value of `read_data` in a variable called `data`.

You don't need to change anything in this cell, but read and make sure you understand what the lines are doing. It's not important for you to understand the code inside these functions right now – we'll discuss them in a coming week.

[2]:
```
url = "https://philchodrow.github.io/PIC16A/homework/HW3-hamilton-data.csv"

fname = "data.csv"
retrieve_data(url, fname)
data = read_data(fname)
```

1.1.2 Part (b): Define `describe` in utils.py

This would also be a good time to inspect the data to make sure you understand how it is structured. Write a function `describe` that describes the meaning of the `n`th row of `data`. Running `describe(data, 5)` on the Hamilton data set should print the following:

```
"Element 5 of the Hamilton data set is ('burr', 'betsy'). This means that Burr mentions Betsy
```

Please attend to capitalization and formatting. While the standard string concatenation operator `+` is completely fine for this task, the fancy `str.format()` function may make your code somewhat simpler. This page has some useful examples in case you'd like to try this.

[3]:
```
# test your describe function here
describe(data, 5)
```

```
Element 5 of the Hamilton data set is ('burr', 'betsy'). This means that Burr
mentions Betsy in a song.
```

1.1.3 Part (c): Define `data_to_dictionary` in utils.py

Write a function called `data_to_dictionary` that converts the data into a dictionary such that:

1. There is a single key for each character in Hamilton.
2. The value corresponding to each key is a list of the characters/airports to which that key links. The list should contain repeats if there are multiple links.

Here's an example of the desired behavior on a fake data set.

```
data = [("a", "b"),
        ("a", "b"),
        ("a", "c"),
        ("b", "c"),
        ("b", "a")]

data_to_dictionary(data)

# output
{"a" : ["b", "b", "c"], "b" : ["a", "c"]}
```

[4]:
```
# run this code to test

toy_data = [("a", "b"), ("a", "b"), ("a", "c"), ("b", "c"), ("b", "a")]
D = data_to_dictionary(toy_data)
print(D)
```

```
{'a': ['b', 'b', 'c'], 'b': ['a', 'c']}
```

1.2 Problem 2: Define `PageRankDiGraph` class in PageRank.py

A **directed graph**, or DiGraph, is just a set of arrows ("edges") between objects ("nodes"). It is a natural way to represent data that represents one-way relationships, such as links from one

webpage to another or mentions of one character by another. We already saw a directed graph above when we introduced the idea of PageRank. Here's a paired-down example.

Example of a directed graph.

Implement a `PageRankDiGraph` class according to the following specs. https://docs.python.org/3/reference/datamodel.html#special-method-names might be helpful. Let `G`, `G1`, `G2` be instances of `PageRankDiGraph`.

- The `__init__` method should accept one argument, `data`, which you can expect to be a list of tuples like our Hamilton data. It should save `data` as `self.data`, then construct an instance variable `self.link_dict` which is simply the output of `data_to_dictionary` applied to the argument `data`.
- `get_nodes` returns a list of all nodes in the graph.
- Running `{character name} in G` should return `True` if the character is a node in the graph, `False` otherwise.
- `print(G)` should print `PageRankDiGraph with {number of nodes} nodes and {number of edges} edges.`
- `G1 + G2` returns a new instance of `PageRankDiGraph` that contains both the edges of `G1` and `G2`.
- Define a method `self.linked_by(x)` which, when called, returns the list of characters that `x` connects to. Hint: It should be a one liner using `self.link_dict`.

Example:

```
D = PageRankDiGraph(data)
print(D.linked_by('peggy'))

# output
['peggy', 'schuylerSis']
```

[5]:
```
G = PageRankDiGraph(data)

# test different methods of PageRankDiGraph here
print(G.get_nodes()) # test get_nodes()
print("peggy" in G) # test __contains__
print(G) # test __str__

G1 = PageRankDiGraph(data)
print(G1 + G) # test __add__

print(G.linked_by("peggy")) # test linked_by(x)
```

```
['generalMercer', 'betsy', 'company', 'philipS', 'men', 'jefferson', 'jAdams',
'philipH', 'hamilton', 'reynolds', 'marthaWashington', 'schuylerSis', 'sally',
'pendleton', 'burr', 'ness', 'lafayette', 'seabury', 'ensemble', 'peggy',
'angelica', 'generalMontgomery', 'sAdams', 'women', 'laurens', 'weeks',
'theodosiaMother', 'admiralHowe', 'theodosiaDaughter', 'rochambeau', 'maria',
'knox', 'madison', 'franklin', 'conway', 'washington', 'lee', 'paine',
'kingLouis', 'kingGeorge', 'mulligan', 'eacker', 'doctor', 'eliza', 'jay',
'green']
```

```
True
PageRankDiGraph with 46 nodes and 293 edges.
PageRankDiGraph with 46 nodes and 586 edges.
['peggy', 'schuylerSis']
```

1.3 Problem 3: Define `PageRankIterator` class in PageRank.py

Define a `PageRankIterator` class that iterates through a `PageRankDiGraph` via a custom `__next__` method.

When initialized, this class should create instance variables to store: - `graph`, the `PageRankDiGraph` instance, given as input - `iteration_limit`, an integer given as input, - `jump_prob`, a number between 0 and 1 (inclusive), given as input, - a counter `iter`, starting at 0, to log the number of steps taken, - `current_state` variable whose value is one of the keys of the `link_dict` of the `PageRankDiGraph`. You can choose its initial value arbitrarily; in my code I chose `self.current_state = "hamilton"`.

Your `__init__` method should check that the input `graph` is an instance of `PageRankDiGraph`, and raise `TypeError` if it is not. Hint: which one of these would be useful? https://docs.python.org/3/library/functions.html

We are going to use iteration to implement the PageRank algorithm. This means we are going to imagine a surfer who is following the links in our data set. **Implement the following two methods:**

1. `follow_link()`.
    1. Pick a random new character mentioned by the current character, or new airport served by the current airport. Let's call this `next_state`.
    2. If `next_state != current_state`, set `current_state` to `next_state`.
    3. Otherwise (if `next_state == current_state`), teleport (see below).
    4. You might run into `KeyError`s, in which case you should again teleport (use a `try-except` block).
2. `teleport()`.
    1. Set the current state to a new state (key of the link dict) completely at random.

Hint: use `random.choice` from the `random` module to choose elements of lists.

Finally, **implement** `__next__()`. `__next__()` should do `follow_link()` with `jump_prob`% probability, and do `teleport()` with `1-jump_prob`% probability. Then return the `current_state`. You should also define a custom `StopIteration` condition to ensure that only as many steps are taken as the `iteration_limit` supplied to the `PageRankDiGraph` initializer.

1. To do something with 85% probability, use the following:

```
if random.random() < 0.85:
    # do the thing
else:
    # do the other thing
```

Example Usage After you define your class, run the following code and show that it works. Note: your precise sequence may be different from mine.

```
I = PageRankIterator(G, 5, 0.6)
for x in I:
    print(x)

following link : current state = burr
following link : current state = washington
following link : current state = burr
following link : current state = hamilton
teleporting    : current state = washington
```

I have added printed messages here for you to more clearly see what should be happening, but it is not necessary for you to do this. It is sufficient for your output to look like:

```
burr
washington
burr
hamilton
washington
```

[6]:
```
# test PageRankIterator here
I = PageRankIterator(G, 5, 0.6)
for x in I:
    print(x)
```

```
reynolds
women
schuylerSis
jay
doctor
```

1.4 Problem 4: Implement class `IterablePageRankDiGraph` from scratch in PageRank.py.

`IterablePageRankDiGraph` is a subclass of `PageRankDiGraph`, and should inherit all of `PageRankDiGraph`'s methods. You need to define three methods, `__init__`, `__str__`, and `__iter__` such that:

- At `__init__`, in addition to what initializing `PageRankDiGraph` requires, it should also get `iteration_limit` (default 20) and `jump_prob` (default 0.75), and save corresponding instance variables.
- If `IG` is an instance of `IterablePageRankDiGraph`, `print(IG)` should print `IterablePageRankDiGraph with {number of nodes} nodes and {number of edges} edges.`
- `__iter__` returns a new instance of `PageRankIterator` initialized with the `IterablePageRankDiGraph` instance, and its `iteration_limit` and `jump_prob` values.

If successful, the following cell should run without throwing any errors.

[7]:
```
from PageRank import IterablePageRankDiGraph
```

```
# test IterablePageRankDiGraph here
Iter = IterablePageRankDiGraph(data)
print(Iter) # test __str__
print(Iter.__iter__()) # test __iter__
```

```
IterablePageRankDiGraph with 46 nodes and 293 edges.
<PageRank.PageRankIterator object at 0x7fe44000c490>
```

1.5 Problem 5

1.5.1 Part (a): Compute PageRank

Finally, we are ready to compute the PageRank in our data set. Initialize a `PageRankDiGraph` with a large iteration limit (say, 1,000,000). We will let our surfer randomly move through the data set this many times. The number of times that the surfer visits state `x` is the PageRank score of `x`.

Create a `dict` which logs how many times a given state appears when iterating through the `PageRankDiGraph`.

Your Solution

[8]:
```
# write your code here
g = IterablePageRankDiGraph(data, 1000000)
D = {} # create a dict which logs how many times a given state appears

for i in g:
    D[i] = 0 # get all the keys from the states

for i in g:
    D[i] += 1 # add the count by 1 every time the state appears

print(D)
```

```
{'eliza': 33517, 'washington': 58258, 'knox': 15589, 'maria': 14659, 'madison':
24290, 'jefferson': 45879, 'hamilton': 101624, 'kingLouis': 14486, 'peggy':
15374, 'seabury': 13441, 'philipS': 18186, 'jAdams': 32795, 'ensemble': 13701,
'kingGeorge': 21422, 'theodosiaDaughter': 14524, 'sAdams': 15882, 'admiralHowe':
14148, 'men': 13666, 'angelica': 31961, 'schuylerSis': 25366, 'theodosiaMother':
14323, 'generalMercer': 14266, 'franklin': 14548, 'philipH': 18884, 'burr':
61852, 'doctor': 13636, 'jay': 14507, 'mulligan': 15822, 'conway': 14774,
'marthaWashington': 14642, 'sally': 15106, 'ness': 14559, 'generalMontgomery':
14406, 'reynolds': 20461, 'company': 13610, 'paine': 14860, 'lafayette': 23027,
'green': 15614, 'weeks': 14545, 'betsy': 14606, 'rochambeau': 15650, 'laurens':
19234, 'lee': 22623, 'pendleton': 14420, 'women': 13735, 'eacker': 17522}
```

1.5.2 Part (b): Display Your Result

Use your favorite approach to show the results in sorted format, descending by PageRank score. The entries at the top should be the entries with highest PageRank. You may show either the complete list or just the top 10.

Consider using the sort method of list https://docs.python.org/3/library/stdtypes.html#list.sort with an appropriate lambda expression - `L.sort(key = lambda ...)`

Check your code by comparing your top 10 to mine. Because we are using a randomized algorithm, your results will not agree exactly with mine, but they should be relatively close. If your top 10 list is very different, then you might want to revisit your previous solutions.

My top 10 were:

```
[('hamilton', 101962),
 ('burr', 61948),
 ('washington', 57775),
 ('jefferson', 45480),
 ('jAdams', 32970),
 ('eliza', 32941),
 ('angelica', 31817),
 ('schuylerSis', 25556),
 ('madison', 24702),
 ('lafayette', 23070)]
```

What are the most important elements in the data set? Does it change with different jump probabilities in the iterator? Does it change with different initial states?

[9]:
```
# write your code here
L = list(D.items())
L.sort(key = lambda x:x[1], reverse = True)
print("My top 10 were: \n")
for i in range(10):
    print("{0}".format(L[i]))
```

```
My top 10 were:

('hamilton', 101624)
('burr', 61852)
('washington', 58258)
('jefferson', 45879)
('eliza', 33517)
('jAdams', 32795)
('angelica', 31961)
('schuylerSis', 25366)
('madison', 24290)
('lafayette', 23027)
```