HW3

Fill in your name and the names of any students who helped you below.

> I affirm that I personally wrote the text, code, and comments in this homework assignment.

- Austin Wuthrich 4/21/22

PageRank

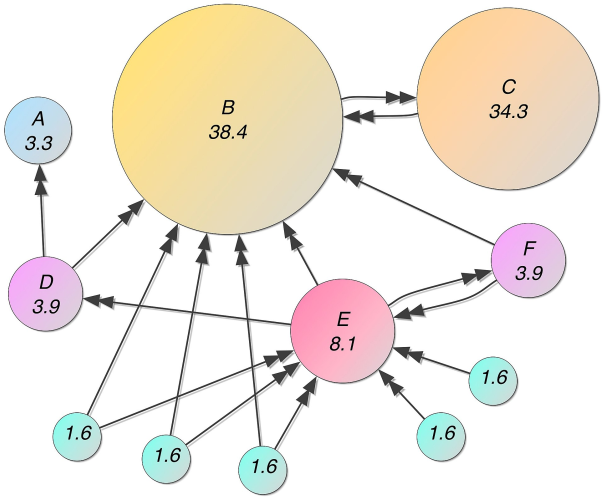
What is the most important website on the internet? Who is the "key player" on a sports team? Which countries are the most central players in the world economy? There is no one correct answer to any of these questions, but there is a most profitable one. PageRank is an algorithm for ranking individual 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.

Comments are required for all parts of this assignment

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. Eventually, the surfer will visit all the pages one or more times. Pages that the surfer visits more frequently have higher *PageRank scores*. 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.



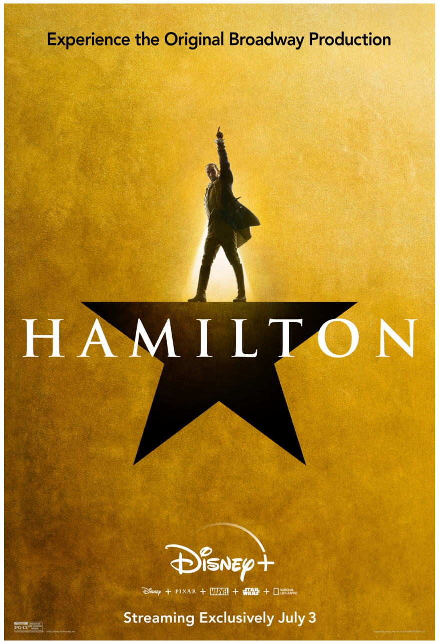
A schematic for PageRank.

Data

You can complete this assignment using data from one of two sources.

Option 1: Hamilton

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

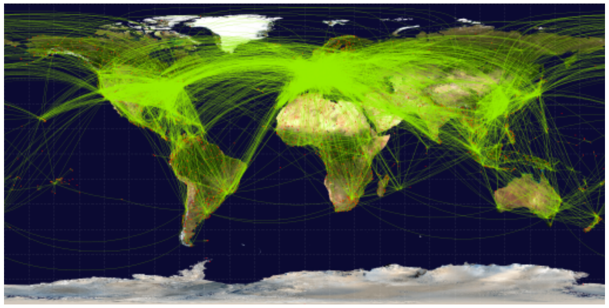

Experience the Original Broadway Production
HAMILTON
Disney+
Streaming Exclusively July 3

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.

If you use this data set, listening to the soundtrack while working is strongly recommended.

Option 2: Global Airline Network



The global airline network

Back in the Before Times, lots of people flew on airplanes. This data set includes a "link" from Airport A to Airport B whenever there is a flight from B to A. This data set was collected by the OpenFlights Project.

(A). Define Functions

In this part, all you have to do is hit `shift + enter` on the code block supplied below. This block defines two functions. The first one retrieves the data from the internet and saves it to your local computer, while the second reads in the data, producing a list of tuples. It's not important for you to be familiar with the code in these functions right now -- we'll discuss them soon.

In [1]:

```
import urllib
import csv

def retrieve_data(url):
    """
    Retrieve a file from the specified url and save it in a local file
    called data.csv. The intended values of url are:

    1. https://philchodrow.github.io/PIC16A/homework/HW3-hamilton-data.csv
    2. https://philchodrow.github.io/PIC16A/homework/HW3-flights-data.csv
    """

    # grab the data and parse it
    filedata = urllib.request.urlopen(url)
    to_write = filedata.read()

    # write to file
    with open("data.csv", "wb") as f:
        f.write(to_write)

def read_data(path):
    """
    read downloaded data from a .csv file, and return a list of tuples.
    each tuple represents a link between states.
    """
    with open(path, "r") as f:
        reader = csv.reader(f)
        return [(row[0], row[1]) for row in list(reader)]
```

(B). Grab the Data

The data live at the following URLs:

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

- **Airline**: `https://philchodrow.github.io/PIC16A/homework/HW3-flights-data.csv`

In each data set, each row corresponds to a "link" between objects. In Hamilton, the pairs have format `mentioner, mentioned` while in the airline network the rows have format `destination, origin`.

Pick one of these data sets, and set the variable `url` appropriately by uncommenting one of the two lines below. Then, call `retrieve_data()` and `read_data()`. The `path` argument for `read_data()` should be set to `"data.csv"`. Create a variable called `data` to hold the return value of `read_data()`.

Your solution

In [2]:

```
# uncomment the second line if you'd prefer to
# work with the flights data.
url = "https://philchodrow.github.io/PIC16A/homework/HW3-hamilton-data.csv"
# url = "https://philchodrow.github.io/PIC16A/homework/HW3-flights-data.csv"

# Call your functions below
retrieve_data(url)
data = read_data("data.csv")
```

(C). Examine the structure of the data

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

```
describe(5)

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

In context of the airline flights data, your function should instead do this:

```
describe(5)

# output
"Element 5 of the flights data set is ('SIN', 'BKK'). This means that there is a flight from BKK to SIN."
```

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.

Your Solution

In [3]:

```
def describe(n):
    """
    Function that inputs index number and textually describes what the Hamilton data for that entry means
    param n: integer index number within range of the data
    returns: no value returned. Function prints message out to console when called.
    """
    #message using string literal formating by position. Accessed values by indexing.
    #Used string method .capitalize() to match example string
    message ='Element {0} of the Hamilton data set is {1}. This means that {2} mentions {3} in a song.'.format(n,data[n],

    print(message) #prints to console
describe(5) #example
```

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

(D). Data to Dictionary

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) or airport (in flights).
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"]}
```

Your Solution

In [4]:

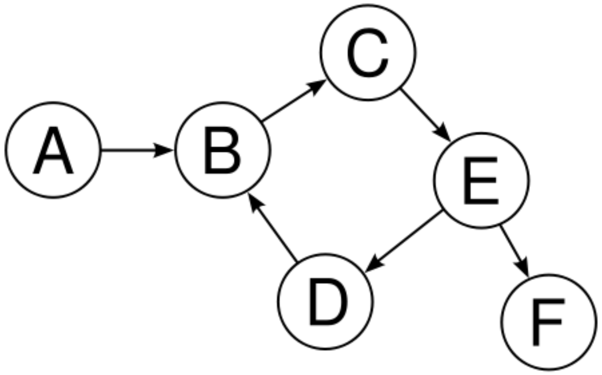
```
def data_to_dictionary(data):
    """
    Function that inputs list of data with each entry a tuple and creates a dictionary with keys
        corresponding to each unique Hamilton character and values that are lists of the characters
        they reference
    Param data: Hamilton data list of tuples representing chaacter connections
    Return: dictionary with unique characters as keys and values that are a list of who they reference
    """
    D={}                        #D declared and initialized as empty


    for i in range(len(data)):  #iterates through length of data
        key = data[i][0]        #accesses key in tuple located at index i of data
        val_str = data[i][1]    #accesses value in tuple located at index i of data
        val_list_at_i = []      #initializes+declares empty values list for each iteration

        if key in D.keys():     #checks if key already exists in D
            val_list_at_i = D.get(key) #if so, assigns value list to val_list_at_i for given key
            val_list_at_i.append(val_str) #appends string value from tuple
            D.update({key:val_list_at_i}) #updates dictionary with new value list

        else:                   #for case where key not in D
            val_list_at_i.append(val_str) #appends string value from tuple to value list
            D.update({key:val_list_at_i}) #updates dictionary with new key and value

    return D                    #returns created dictionary
data_to_dictionary(data)        #example
```

Out[4]:

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

(E). Define a PR_DiGraph class

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 `PR_DiGraph` class with a custom `__init__()` method and a `linked_by()` method. The `__init__()` method should accept two arguments: `data` and `iteration_limit` . The `__init__()` method should then construct an instance variable `self.link_dict` which is simply the output of `data_to_dictionary` applied to the argument `data` . `__init__()` should also construct an instance variable `self.iteration_limit` , which simply takes the value of `iteration_limit` supplied to `__init__()` . Don't worry about that one for now.

Then, define a method `self.linked_by(x)` which, when called, returns the value `self.link_dict[x]` .

Finally, add an `__iter__` method, which returns an object of class `PR_Iterator` . We will define this class in the next part.

Example session (using Hamilton):

```
D = PR_DiGraph(data, iteration_limit = 10000)
D.linked_by('peggy')

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

Your Solution

In [5]:

```
class PR_DiGraph():
    """
    Class that represents a directed (one way) graph between nodes (characters) in the Hamilton data set
    """

    def __init__(self, data, iteration_limit): #initializer for instance variables
        self.link_dict = data_to_dictionary(data) #initalizes linked dictionary of Hamilton data
        self.iteration_limit = iteration_limit    #initializes number of times to perform surfing btwn nodes

    def linked_by(self,x):
        """
        Function that returns who in the data set is linked to whom
        param self: the instance of the directed graph class calling the function
        param x: the name of the Hamilton character that the function looks for characters linked to
        returns: list of characters in the data set that the character is connected to
        """
        return self.link_dict[x] #Returns list of characters linked to based on character in argument

    def __iter__(self):          #Iterator that allows movement through nodes
        return PR_Iterator(self) #Returns class that performs iteration to other nodes

D = PR_DiGraph(data, iteration_limit = 10000) #example declaration of a node D
D.linked_by('peggy')  #example method and input
```

Out[5]:

```
['peggy', 'schuylerSis']
```

(F). Implement PR_Iterator

Define a `PR_Iterator` class with a custom `__next__()` method.

The `__init__` method of this class should create instance variables to store the `PR_DiGraph` object from which it was constructed; a counter `i`, starting at 0, to log the number of steps taken, and a `current_state` variable whose value is one of the keys of the `link_dict` of the `Pr_DiGraph`. You can choose its initial value arbitrarily; in my solution code I chose `self.current_state = "hamilton"`.

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()`.
   A. Pick a random new character mentioned by the current character, or new airport served by the current airport. Let's call this `next_state`.
   B. If `next_state != current_state`, set `current_state` to `next_state`.
   C. Otherwise (if `next_state == current_state`), teleport (see below).
   D. You might run into `KeyError`s, in which case you should again teleport (use a `try-except` block).
2. `teleport()`.

A. 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 85% probability, and do `teleport()` with 15% probability. You should also define a custom `StopIteration` condition to ensure that only as many steps are taken as the `iteration_limit` supplied to the `PR_DiGraph` 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.

```
D = PR_DiGraph(data, iteration_limit = 5)
for char in D:
    print(char)
```

```
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:

```
D = PR_DiGraph(data, iteration_limit = 5)
for char in D:
    print(char)
```

```
burr
washington
burr
```

```
hamilton
washington
```

Your Solution

In [6]:

```
import random          #for link-surfing later

class PR_Iterator:
    """
    Class that iterates through the links from the character data in the PR_DiGraph
    """
    def __init__(self, PR_DiGraph):
        self.PR_DiGraph = PR_DiGraph
        self.i=0                      #initializes start counter for iteration
        self.current_state="hamilton" #default current state node "hamilton"
        self.iteration_limit = PR_DiGraph.iteration_limit #initializes iteration limit from PR_DiGraph

    def teleport(self):
        """
        Class method that jumps the class to another node at random
        """
        dict_keys = list(self.PR_DiGraph.link_dict.keys()) #creates list of character keys in Hamilton data dictionary
        self.current_state = random.choice(dict_keys)      #assigns current state to random character in keys
        return self.current_state                          #returns new node location

    def follow_link(self):
        """
        Class method that allows PR_Iterator to travel connections to different nodes
        """
        try:   #attempts although possibility of KeyError from small list of options for linked characters to current sta
            linked_char_list = self.PR_DiGraph.linked_by(self.current_state) #list of characters linked to current charac
            self.next_state = random.choice(linked_char_list) #selects random connected character to move to
            if (self.next_state != self.current_state): #checks if random selected character is different than current ch
                self.current_state = self.next_state    #if so, updates
            else:                                       #escapes dead end
                self.teleport()                         #through teleportation
        except KeyError:                                #addresses occasional error from lack of options at a node
            self.teleport()                             #through teleporation
        return self.current_state                       #returns new current state

    def __next__(self):                                 #directs iterator class where to go next
        self.i+=1                                       #incrimentation of counter every time next() called
        if (self.i <= self.iteration_limit):            #termination condition: checks if below iteration limit
            if random.random() < 0.85:                  #85% of time will call follow_link()
```

```
                return self.follow_link()
            else:                                       #15% of time calls teleport()
                return self.teleport()
        else:                                           #termination of iteration
            raise StopIteration
```

In [16]:

```
# run the below
D = PR_DiGraph(data, iteration_limit = 5)
for char in D:
    print(char)
```

```
lafayette
burr
eliza
philipH
philipS
```

(G). Compute PageRank

Finally, we are ready to compute the PageRank in our data set. Initialize a `PR_DiGraph` with a large iteration limit (say, 1,000,000). Use a `for` -loop to allow your surfer to randomly move through the data set. 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 `PR_Digraph` . So, this dictionary holds the PageRank score of each state.

Your Solution

In [10]:

```
Ham = PR_DiGraph(data, iteration_limit = 1000000)  #example initialization of PR_DiGraph class
PageRank = {}                       #Dictionary storing characters and frequency of visits over iteration

for char in Ham:                    #iterates over class
    key = char                      #constructs dictionary with key being the character
    if key in PageRank.keys():      #checks if key already exists, if so
        val = PageRank.get(key)     #extracts value from dictionary
        val+=1                      #incriments by 1
        PageRank.update({key:val})  #updates dictionary with new key:val
    else:                           #Otherwise, adds new key and val into dictionary
```

```
        val = 1
        PageRank.update({key:val})

print(PageRank)                       #Unsorted PageRank dictionary example
```

```
{'washington': 92636, 'burr': 99000, 'philipS': 7869, 'company': 17034, 'angelica': 48019, 'schuylerSis': 19245, 'madiso
n': 37164, 'hamilton': 166237, 'laurens': 27675, 'eliza': 52309, 'reynolds': 29316, 'lee': 33263, 'admiralHowe': 658, 'do
ctor': 17094, 'philipH': 26231, 'mulligan': 21272, 'women': 16879, 'jefferson': 72195, 'ensemble': 17282, 'jAdams': 3126
0, 'generalMercer': 1694, 'seabury': 16828, 'men': 17192, 'eacker': 6131, 'conway': 1726, 'lafayette': 34308, 'kingGeorg
e': 28793, 'ness': 1699, 'knox': 3906, 'peggy': 20311, 'sally': 2777, 'betsy': 1680, 'green': 3982, 'sAdams': 3331, 'king
Louis': 1754, 'paine': 1961, 'weeks': 1672, 'maria': 1639, 'franklin': 1946, 'rochambeau': 3970, 'jay': 1680, 'theodosiaD
aughter': 1650, 'pendleton': 1675, 'generalMontgomery': 1647, 'theodosiaMother': 1676, 'marthaWashington': 1734}
```

(H). 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. What are the most important elements in the data set?

You may show either the complete list or just the top 10.

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.

For Hamilton, my top 10 were:

```
[('hamilton', 166062),
 ('burr', 99180),
 ('washington', 92246),
 ('jefferson', 72450),
 ('eliza', 51485),
 ('angelica', 48042),
 ('madison', 37421),
 ('lafayette', 34297),
 ('lee', 33678),
 ('jAdams', 31121)]
```

For the flights data, my top 10 were:

```
[('LHR', 18043), # London Heathrow
 ('ATL', 16370), # Atlanta
 ('JFK', 14795), # New York JFK
 ('FRA', 14156), # Frankfurt
 ('CDG', 14073), # Charles de Gaulle (Paris)
 ('LAX', 13199), # Los Angeles
 ('ORD', 12915), # Chicago O'Hare
 ('PEK', 12525), # Beijing
 ('AMS', 12410), # Amsterdam Schiphol
 ('PVG', 11517)] # Shanghai
```

Your solution

In [23]:

```
#Dictionary comprehension that sorts each item (key:value) based off the PageRank values in descending order
sorted_PageRank = {key:val for key,val in sorted(PageRank.items(), key=lambda x:x[1], reverse=True)}

print(sorted_PageRank)  #Example of PageRanks for Hamilton script based off my test run
```

```
{'hamilton': 166237, 'burr': 99000, 'washington': 92636, 'jefferson': 72195, 'eliza': 52309, 'angelica': 48019, 'madiso
n': 37164, 'lafayette': 34308, 'lee': 33263, 'jAdams': 31260, 'reynolds': 29316, 'kingGeorge': 28793, 'laurens': 27675,
'philipH': 26231, 'mulligan': 21272, 'peggy': 20311, 'schuylerSis': 19245, 'ensemble': 17282, 'men': 17192, 'doctor': 170
94, 'company': 17034, 'women': 16879, 'seabury': 16828, 'philipS': 7869, 'eacker': 6131, 'green': 3982, 'rochambeau': 397
0, 'knox': 3906, 'sAdams': 3331, 'sally': 2777, 'paine': 1961, 'franklin': 1946, 'kingLouis': 1754, 'marthaWashington': 1
734, 'conway': 1726, 'ness': 1699, 'generalMercer': 1694, 'betsy': 1680, 'jay': 1680, 'theodosiaMother': 1676, 'pendleto
n': 1675, 'weeks': 1672, 'theodosiaDaughter': 1650, 'generalMontgomery': 1647, 'maria': 1639, 'admiralHowe': 658}
```

(I). Submit!

Check that your code is appropriately documented (comments and docstrings), and turn it in.