Homework 6: DFS + Page Rank

As with all assignments, make sure that you read, understand, and follow the relevant course policies (https://canvas.rice.edu/courses/52701/assignments/syllabus) (honor code, late policy, etc.)!

This assignment is due at **5:00pm CST on Monday, November 21st, 2022.** As per the course grading policy, late submissions will be accepted until **5:00pm CST on Wednesday, November 23rd, 2022** for partial credit. After that date, no further submissions will be accepted for credit.

1. The Goal

This is the third and final assignment in which you will analyze data extracted from Wikipedia. In homework 4, you used graphs to model and explore the relationships between articles. In this assignment, you will build upon the work that you did in homework 4, and will implement more algorithms that operate on graphs.

1.A. Pedagogical Goals

In completing this assignment, you will continue practicing many of the concepts from homeworks 1-5, and will additionally gain experience with:

1. Making minor modifications to an algorithm in order to generalize it.
2. Designing and implementing recursive functions.
3. Understanding and implementing the page rank algorithm.

Make sure to read the entire assignment before you begin!

2. Before You Begin

Once again, you'll be developing your code locally using PyCharm. Therefore, you will need to download the provided code and data files for this assignment, which can be found in **this zip file (https://canvas.rice.edu/courses/52701/files/3791958/download)**. Once you've downloaded this file, you should unzip it into a directory of your choice. Within the unzipped folder, you will find the following files:

- **comp614_module6.py:** This contains several class definitions: Stack, Queue, and DiGraph. It also contains my implementation of the file_to_graph method from homework 4, which you will need to use for testing and analysis.
- **hw6.py:** This is the template code for the functions that you will implement in this assignment. Included in this file is my implementation of the BFS algorithm from homework 4 (which is now called "bfs_dfs").
- **wikipedia_articles_streamlined.txt:** This is a text file that includes a real data set that was mined from Wikipedia, in the same format that you encountered in homework 4. It is a streamlined version of the file that you were given in homework 4, with only 785 nodes. This

streamlining was performed in order to allow you to run the PageRank algorithm efficiently on this graph.
- **test0.txt - test9.txt:** These are text files that include small artificial data sets that you can use for testing, in the same format that you encountered in homework 4.

3. Your Task (Code + Writeup)

You should work in the provided template, **hw6.py**. As you implement each function, you should test your work using **OwlTest (https://py3.owltest.org/owltest/?urlTests=comp614.hw6_tests.py&urlPylintConfig=comp614.pylint_config.py&imports=%7Bcomp614:comp614_module6%7D)**. As in the previous assignments, you'll need to upload the latest version of your hw6.py file using OwlTest's "upload a local Python file" capability by pressing the "Choose File" button and selecting the correct file from your filesystem.

3.A. DFS (Code + Writeup)

The first algorithm that you will be implementing in this assignment is DFS. You will be implementing two different versions of this algorithm: one that is iterative, and one that is recursive.

First, you will need to modify the provided bfs_dfs so that you can use the same function to implement both BFS and DFS. This function should take the following inputs:

- **graph:** The directed graph on which to perform the search.
- **start_node:** The node from which to begin the search.
- **rac_class:** The type of "restricted access container" to be used to perform the search. This will be a reference to either the provided Queue class, or the provided Stack class. When rac_class is Queue, your function should instantiate an instance of the provided Queue class and perform BFS; when rac_class is Stack, your function should instantiate an instance of the provided Stack class and perform DFS. **We will manually verify that you are using the correct structure in each case, and deductions will apply if you're using the wrong structure!**

You should modify the provided function such that it uses the appropriate type of RAC and only returns the parent mapping. You can still keep and use the distance mapping within your function if you'd like, but you should not return it.

Next, you will need to implement a recursive version of DFS. You need to write a recursive function, recursive_dfs, that takes the following inputs:

- **graph:** The directed graph on which to perform the search.
- **start_node:** The node from which to begin the search.
- **parent:** A dictionary mapping nodes that have already been explored to their parents. When recursive_dfs is initially called by OwlTest, the dictionary that's passed as an input will contain only contain {start_node: None}. However, within your implementation of recursive_dfs, you should modify this parent mapping appropriately.

Note that recursive_dfs does not need to return anything, and OwlTest will not check its output; instead, OwlTest will have a reference to the input parent dictionary, and will check at the end to see if that input dictionary has been modified appropriately.

Your implementation of recursive_dfs must be recursive. It may not call bfs_dfs, and each individual invocation of dfs may only look at the neighbors of one node in the input graph. We will be manually checking this during grading, and if your implementation does not meet this criteria, you will lose credit.

Since recursion is a relatively new and challenging concept, you must complete the following pre-write questions and include your answers in your writeup. This should help you to brainstorm your solution!

1. **Question 3.A.i:** What is/are the base case(s) for this function? For each base case, you should clearly identify the following:
   - The **when**: The condition(s) under which we fall into this base case. You can & should refer to any parameters used by name.
   - The **what**: A detailed explanation of what needs to happen when we're in this base case.You can & should refer to any parameters used by name.
2. **Question 3.A.ii:** What is/are the recursive case(s) for this function? For each recursive case, you should clearly identify the following:
   - The **when**: The condition(s) under which we fall into this recursive case.You can & should refer to any parameters used by name.
   - The **what**: A detailed explanation of what needs to happen when we're in this recursive case. You can & should refer to any parameters used by name. You are also welcome to assign names to local variables that you intend to define. For any recursive calls that you intend to make, you should clearly specify what the inputs will be in terms of these parameters and local variables.

After answering these questions, you should be ready to implement dfs!

3.B. PageRank (Code + Writeup)

The second algorithm that you will be implementing in this assignment is the PageRank algorithm. As discussed in the video lectures, this algorithm gives us a metric for each page approximating its relative "importance". This can alternatively be thought of as a metric measuring the likelihood of landing on a particular page after performing a random walk on the graph.

In this assignment, will need to implement three functions related to the PageRank algorithm. First, you will need to implement a function called get_inbound_nbrs, which should take as its input a directed graph and return a dictionary mapping each node n in the graph to the set of nodes from which there are edges pointing to n.

Second, you will need to implement a function remove_sink_nodes, which should take as its input a directed graph. This function should return a new copy of the graph, with minor modifications. Specifically, it should find every sink node -- where a sink node is defined as a node with no

outbound edges -- and add an edge from that node to every other node in the graph, excluding itself. This function should return the new version of the graph.

Finally, you will need to implement a function called page_rank that takes as its input a directed graph and a real number representing the damping factor. You can assume that the damping factor will be between 0 and 1, inclusive. This function should implement the page_rank algorithm according to the high-level description found **here (https://canvas.rice.edu/courses/52701/pages/pagerank-algorithm)**. However, before you implement the page_rank function, you should answer the following pre-write questions, which are designed to ensure that you understand the algorithm. Include your answers in your writeup. Be as specific as possible in your answers!

1. **Question 3.B.i:** Assume that you are not using any list comprehensions in your implementation. What is the minimum "depth" that your loop nest will need to be in order to implement this algorithm? Why? You should clearly describe the purpose of each loop.
   - You can think of depth in terms of the level of indentation. For example, the following code has a depth of four:

```
for idx1 in range(5):
    for idx2 in range(10):
        for idx3 in range(20):
            for idx4 in range(40):
                ## Do something
```

   - On the other hand, the following code involves three loops, but only has a depth of two:

```
for idx1 in range(5):
    for idx2 in range(10):
        ## Do something
    for idx3 in range(20):
        ## Do something
```

2. **Question 3.B.ii:** How will you ensure that you are updating all of the page ranks "simultaneously"? In other words, how will you ensure that you are using the old $PageRank_{k-1}$ values rather than some mix of old and new values in order to compute the new $PageRank_k$ values?
3. **Question 3.B.iii:** If we think of the page ranks in terms of a random walk on the graph, what does the damping factor, d, represent? If we were to set d to 1, in what way(s) could that adversely impact the page ranks computed by the algorithm, and why? If we were to set d to 0, in what way(s) could that adversely impact the page ranks computed by the algorithm, and why?
4. **Question 3.B.iv:** What does $delta_k$ represent? Why do we want to stop when $delta_k < 10^{-8}$?

4. Writeup & Submission (Code + Writeup)

This assignment needs to be submitted in two parts. First, you will need to submit your code to the "Homework 6: DFS + PageRank (Python)" assignment on Canvas. You should see a button that says "Load Homework 6: DFS + PageRank (Python) in new window"; clicking on that should take you to CanvasTest, which should have a **white background** (as opposed to OwlTest's

yellow background). Submit your code through CanvasTest, and **make sure that you can see your grade in the gradebook on Canvas. If you don't see a grade in the gradebook, your code has not been submitted and you will receive a zero!**

Second, you will need to submit a writeup to the "Homework 6: DFS + PageRank (Writeup)" assignment on Gradescope. You can access Gradescope via the "Gradescope" link on the lefthand menu. You should submit your writeup as a single **pdf file** . Your writeup should contain the following components:

1. **A CodeSkulptor URL containing the contents of your hw6.py file.** Although you should be developing your code locally, when you're ready to submit you should copy and paste the contents of hw6.py into CodeSkulptor, save it, and include the link in your writeup. We need this in order to check your code for good style (beyond the auto-checking that is done by CanvasTest). If you do not include this, you will receive the maximum deduction for style.
2. **Your answers to the recursive DFS pre-write questions** (from Section 3.A).
3. **Your answers to the PageRank pre-write questions** (from Section 3.B).
4. **An answer to the following discussion question:**
    1. Run PageRank on a graph built from wikipedia_articles_streamlined.txt with a damping factor of 0.85. Which ten articles have the greatest page ranks? Provide the top ten articles and their page ranks in the form of a list of tuples of (title, page rank), sorted in descending order. Do the results align with your expectations? Why or why not? Why do you think that these pages had the highest ranks?
5. **A brief reflection on your experience with this assignment.** Please see the **Guide to Homework Reflection (https://canvas.rice.edu/courses/52701/pages/guide-to-homework-reflection)** for details on the content and format of your reflection!

You may submit as many times as you like. I encourage you to submit early and often! However, only your last submission to each part of the assignment will count towards your grade. Further, your last submission will be used to determine whether or not your assignment is late. Late submissions will be subject to the course grading policies laid out in **the syllabus (https://canvas.rice.edu/courses/52701/assignments/syllabus)** .