Project #1 Description

For your first project, you will write Python code that creates dictionaries corresponding to some simple examples of graphs. You will also implement two short functions that compute information about the distribution of the in-degrees for nodes in these graphs. You will then use these functions in the Application component of Module 1 where you will analyze the degree distribution of a citation graph for a collection of physics papers. This final portion of module will be peer assessed.

We will use Python 2 in this class since OwlTest (the machine grader) supports Python 2. For more information on recommended Python IDEs, please consult this class page. Note that this portion of the module should be simple for experienced Python programmers. If you find it challenging, your Python skills may not be sufficient to be successful in this class.

Representing directed graphs

To gain a more tangible feel for how directed graphs are represented as dictionaries in Python, you will create three specific graphs (defined as constants) and implement a function that returns dictionaries corresponding to a simple type of directed graphs. If you are unclear on how to represent a directed graph as a dictionary, we suggest that you review the class notes on graph basics. For this part of the project, you should implement the following:

1. **`EX_GRAPH0`**, **`EX_GRAPH1`**, **`EX_GRAPH2`** - Define three constants whose values are dictionaries corresponding to the three directed graphs shown in these linked diagrams: EX_GRAPH0, EX_GRAPH1, and EX_GRAPH2. Note that the label for each node in the diagrams should be represented as an integer. You should use these graphs in testing your functions that compute degree distributions.

- **`make_complete_graph(num_nodes)`** - Takes the number of nodes **`num_nodes`** and returns a dictionary corresponding to a complete directed graph with the specified number of nodes. A *complete* graph contains all possible edges subject to the restriction that self-loops are not allowed. The nodes of the graph should be numbered `0` to **`num_nodes`** `- 1` when **`num_nodes`** is positive. Otherwise, the function returns a dictionary corresponding to the empty graph.

Computing degree distributions

For the second part of this project, you will implement two functions that compute the distribution of the in-degrees of the nodes of a directed graph.

- **`compute_in_degrees(digraph)`** - Takes a directed graph **`digraph`** (represented as a dictionary) and computes the in-degrees for the nodes in the graph. The function should return a dictionary with the same set of keys (nodes) as **`digraph`** whose corresponding values are the number of edges whose head matches a particular node.
- **`in_degree_distribution(digraph)`** - Takes a directed graph **`digraph`** (represented as a dictionary) and computes the unnormalized distribution of the in-degrees of the graph. The function should return a dictionary whose keys correspond to in-degrees of nodes in the graph. The value associated with each particular in-degree is the number of nodes with that in-degree. In-degrees with no corresponding nodes in the graph are not included in the dictionary.

Note that the values in the unnormalized distribution returned by this last function are integers, not fractions. This unnormalized distribution is easier to compute and can later be normalized to sum to one by simply dividing each value by the total number of nodes.

Grading and coding standards

As you implement each function, remember to test that function thoroughly using test data of your creation. Once you are confident that your implementation is correct, submit your code to this Owltest page. This page will automatically test your project. The feedback from the OwlTests may reference certain pre-defined graphs used during the tests. You can access the specific definitions of these graphs in this file to aid in your debugging.

Note that trying to debug your project using only the tests in OwlTest can be very tedious since they are slow and give limited feedback. For more information on the machine grading process, feel free to review the pre-class video from "Principles of Computing" on "Testing and machine grading".

OwlTest uses Pylint to check that you have followed the coding style guidelines for this class. If you have not taken "Principles of Computing", please review the pre-class video on "Coding styles and standards". Deviations from these style guidelines will result in deductions from your final score. Please read the feedback from Pylint closely. If you have questions, feel free to consult this page and the class forums.

When you are ready to submit your code to be graded formally, submit your code to the CourseraTest page for this project that is linked on the main programming assignment page. Remember that submitting to OwlTest does not record a grade for the assignment.

Mark as completed