Directed graphs

Recall that for digraphs (a.k.a. directed graphs) each edge is oriented from a tail node to a head node. An edge from node `0` to node `1` in a digraph does not imply that an edge exists from `1` to `0`. In fact it is possible for none, either or **both** such edges to exist in a given digraph! We can modify our dictionary representation of graphs to represent digraphs as follows:

To represent an edge from node `0` to node `1` we place `1` in `0`'s adjacency set but, unlike our graph representation, we do not automatically place `0` in `1`'s adjacency set. The adjacency set for a given node key then records only those nodes which are "pointed to" from the key. The statement

```
if 1 in digraph[0]:
```

thus represents the question "Is there an edge from node `0` to node `1`?" Because the adjacency sets of `0` and `1` are no longer symmetric we can now populate them independently to represent only those nodes immediately reachable from a given node.

Exercise 1 - Your First Digraph

Write the Python code to represent a digraph containing the nodes V = {0, 1, 2, 3, 4, 5} where an edge exists from `n` to `n+1` (for n = 0...4). Also include the edge (5,0). Show your code to a lab assistant for verification.

Exercise 2 - In-Degree

Write a function

```
def in_degree(digraph, node):
    """
    Computes the in-degree of the given node.

    Arguments:
    digraph -- a dictionary representation of a digraph.
    node    -- the given to node.

    Returns:
    The in-degree of node.
    """
    ...
```

to compute the in-degree for a node in a given digraph.

Assert your function produces the following output:

```
>>> digraph = {  1 : set([2]),
                 2 : set([3,4]),
```

```
                 3 : set([]),
                 4 : set([3,2]),
                 5 : set() }
>>> in_degree(digraph, 3)
2
```

```
>> in_degree(digraph, 4)
1
```

```
>> in_degree(digraph, 1)
0
```

Created Sat 23 Aug 20149:17 PM BST

Last Modified Sun 24 Aug 2014 5:46 AM BST