Module 11 Homework - Graphs

Overview

Implement data structure dependent methods for a directed, non-weighted graph in `AdjacencySetGraph` and `EdgeSetGraph` classes. Then implement a parent class `Graph` with methods that are independent of the underlying data structures:

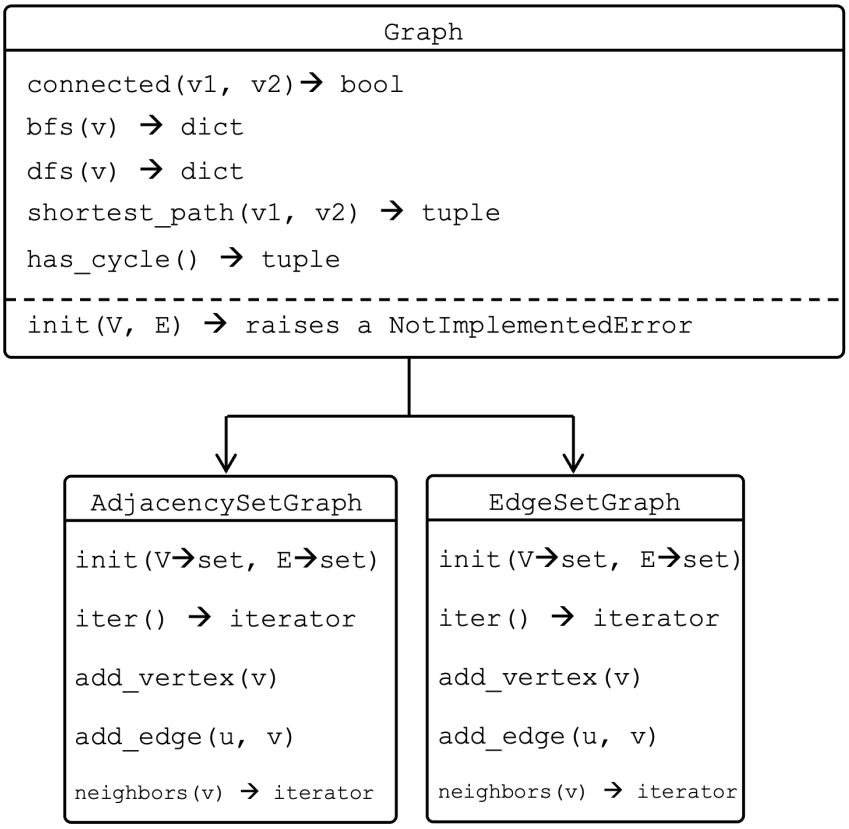

Figure 1: Class diagram showing expected output types for each method. AdjcacencySetGraph and EdgeSetGraph both inherit from Graph.

We use arrows in the figure above to denote typing of parameters and return types (like `bfs()`, which should return a dictionary).

The `Graph` class is a convenient way to factor out common functionality, but should not be used on it's own - users should specify an `AdjacencySetGraph` or `EdgeSetGraph`. We explicitly raise a `NotImplementedError` in `Graph.init` to ensure this.

AdjacencySetGraph

Store a dictionary of `vertex:set(vertices)` pairs, to allow fast iteration over neighbors.

- Inherits from `Graph`
- `__init__(V, E)` - initialize a graph with a set of vertices and a set of edges (we'll use tuples as edges, so E should be a set of tuples). Both parameters should be optional - a user should be able to use `asg = AdjacencySetGraph()` to create an empty graph.

- `__iter__()` - returns an iterator over all **vertices** in the graph
- `add_vertex(v)` - adds vertex to graph
- `add_edge(u, v)` - adds edge `(u, v)` to graph
- `neighbors(v)` - returns an iterator over all out-neighbors of `v`.

EdgeSetGraph

Adhere to and implement all of the above bullets, but use an edge set instead of an adjacency set (store a set of vertices and a set of edges instead of a dictionary of `vertex:set(neighbors)` pairs).

Graph Methods

Methods whose implementations that do not depend on the underlying data structure (though the running times could differ) are factored out here.

- `connected(v1, v2)` - returns True (False) if there is (is not) a path from `v1` to `v2`.
- `bfs(v)` - returns a valid breadth-first search tree (see below for details).
  - *Be careful about efficiency here*: use an efficient queue.
  - Note there could be multiple valid BFS trees based on the same graph. As long as your tree represents a valid BFS traversal, you will receive full credit.
- `dfs(v)` - returns a valid depth-first search tree (see below for details).
  - *Be careful about efficiency here*: use an efficient stack.
  - Note there could be multiple valid DFS trees based on the same graph. As long as your tree represents a valid DFS traversal, you will receive full credit.
- `shortest_path(v1, v2)` - returns a tuple of the minimum distance between `v1` and `v2` and a list containing the edges of a minimal distance path from `v1` to `v2` (the edges should be in the proper order to traverse from `v1` to `v2`). If there is no path from `v1` to `v2`, return `(float("inf"), None)`.
- `has_cycle()` - returns a tuple of True and a list of edges that comprise a cycle if the graph has a cycle (the edges should be in the proper order to traverse the cycle completely). Returns `(False, None)` otherwise.

BFS/DFS Trees

For both `bfs` and `dfs`, you must return a corresponding tree in a dictionary. Refer to lecture slides or Chapter 21 in the textbook for more info.

Some Example Behavior

```
>>> V = {'A', 'B', 'C', 'D', 'E', 'F'}
>>> E = {('A', 'B'), ('A', 'C'),
         ('B', 'C'), ('B', 'D'),
         ('C', 'E'),
         ('D', 'F'),
         ('E', 'D'),
         ('F', 'E')}
```

```
>>> g = AdjacencySetGraph(V, E)

>>> g.connected('A', 'E')
True

>>> g.bfs('A')
{'A': None, 'B': 'A', 'C': 'A', 'D': 'B', 'E': 'C', 'F': 'D'}

>>> g.dfs('B')
{'B': None, 'C': 'B', 'E': 'C', 'D': 'E', 'F': 'D'}

>>> g.has_cycle()
(True, [('D', 'F'), ('F', 'E'), ('E', 'D')])

>>> g.shortest_path('A', 'F')
(3, [('A', 'B'), ('B', 'D'), ('D', 'F')])
```

Imports

No imports allowed on this assignment, with the following exceptions:

- Any modules you have written yourself
- `Queue` from the `queue.py` file distributed with the starter code
- `Stack` from the `stack.py` file distributed with the starter code
- `typing` - this is not required, but some students have requested it

Submission

At a minimum, submit the following files to Gradescope:

- `graph.py`
  - `class Graph`
  - `class AdjacencySetGraph`
  - `class EdgeSetGraph`

Students must submit individually by 11:59 PM EST on Friday, April 26, 2024.

Grading

This assignment will be fully auto-graded on Gradescope.