CSE 4256

Due Sunday Feb 27 11:59 PM EST

Dr. Alan Weide

Spring Term 2022

Homework 6

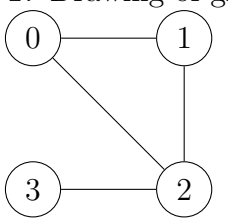
Code Quality

Recently, several homework assignments have been submitted that do not run, and some even have *misspelled keywords*! This is a clear indication that you did not test your code, which in turn is a clear indication that you did not put forth a reasonable effort on the assignment. **Therefore, starting with this assignment, any submitted homework that raises any errors when I invoke Python with your submitted main.py file will receive no credit.**

Background

A (undirected) *graph* is a structure consisting of a set of *vertices* that are connected by *edges*. Formally, a graph G is a pair (V, E) of V, the set of vertices and E, the set of edges (each element of E is a set of two elements in V). For example, the graph $G = \big(\{0, 1, 2, 3\}, \{\{0, 1\}, \{0, 2\}, \{2, 1\}, \{2, 3\}\}\big)$ is drawn below, in fig. 1.

Figure 1. Drawing of graph G.



There are several ways to represent a graph using common data structures available in popular programming languages, each making tradeoffs in performance, memory, and programming complexity.

- As a list of pairs, each of which is an edge in the graph. This is called an "edge list" representation of a graph. In this representation, the set of vertices is the set of unique members in the pairs in the edge list. The graph G from above would be represented by the list `_edgelist = [(0, 1), (0, 2), (2, 1), (2, 3)]`. Observe that the pairs

in `_edgelist` must be interpreted as *unordered* to maintain the undirectedness of the graph.

- As a matrix `m`, wherein the cell `m[i][j]` has the value `True` if $\{i, j\}$ is in the set of edges, and the value `False` otherwise. This is called an "adjacency matrix" representation of a graph. The graph G from above would be represented by the following matrix.

```
_matrix = [[False, True, True, False],
           [True, False, True, False],
           [True, True, False, True],
           [False, False, True, False]]
```

  Observe that for an undirected graph, `_matrix` is diagonally symmetric, that is, for all i and j, `m[i][j] == m[j][i]` (this is easier to see if we replace `True` with `1` and `False` with `0`).

- As a dictionary mapping each vertex to a list of the vertices that are adjacent to it (*i.e.*, reachable in "one hop"). This is called an "adjacency list" represenation of a graph. The graph G from above would be represented by the dictionary `_dict = {0: [1, 2], 1: [0, 2], 2: [0, 1, 3], 3: [2]}`. Observe that if G is an undirected graph, for all i and j, `j` **`in`** `_dict[i]` if and only if `i` **`in`** `_dict[j]`.

A Caveat. To keep all of our representations equivalent, we will make two minor adjustments to the general undirected graph representation problem. First, we assume every vertex is labeled with a nonnegative integer.[1] Second, we will collapse all self-loops. That is, if there is an edge (v, v) in the edge set for a graph, it should be treated as *no edge* when computing the edge set.[2] As you work through this assignment, consider why these two decisions were made and why the footnotes are true.

[1] We do not need to assume the labels are contiguous.

[2] Equivalently, we could say that *every* vertex has an implicit self-loop that is not included in the edge set.

Assignment

This assignment should be completed as a pair of Python source (`.py`) files with the following names:

- `main.py`, modified from the provided `main.py` which contains several test cases (available on Carmen)
- `graph.py` modified from the provided `graph.py` (available on Carmen)

Each question (including the Challenge Activities) has an associated `TODO` comment in the template files.

(1) In the file `graph.py`, there is an abstract base class called `Graph`, with several abstract methods in it. In addition, there are several unimplemented classes: `EdgelistGraph`, `MatrixGraph`, and `DictGraph`. Implement one of these classes. (I recommend starting with `DictGraph`.) See the Background section above for a more detailed description of these representations of a graph.

(2) The `main.py` file available on Carmen contains a simple test script for the initializers of each of the Graph subclasses. Modify `main.py` with additional test cases that exercise the full breadth of behavior of a graph.

Challenge Activities

Some homeworks (such as this one) will have additional challenge activities. These activities **do not contribute to your grade**, but they are problems that I find interesting or challenging.

(3) Does your implementation of the graph class keep a "minimal" representation? That is, does the representation have duplicate edges, vertices, or both? If the representation is not kept minimal, modify your solution with a private `_cleanup(self)` method that's called every time the underlying representation value changes that maintains the invariant the the representation is somehow "minimal" for this graph.

(4) Implement the other two subclasses of `Graph`. **Hint**: a future homework will ask you to implement these other classes.

(5) There are two (non-abstract) functions in the `Graph` class that are marked with `TODO (challenge)` comments: `depth_first_search` and `breadth_first_search`. Implement them as generator functions. **Hint**: a future homework will ask you to implement these generator functions.

(6) A naive implementation of MatrixGraph will not work unless the vertices are labeled with the integers in $[0, |V|)$. Ensure that the `MatrixGraph` implementation accepts and correctly stores graphs with vertices that have arbitrary (nonnegative integer) labels.

(7) This homework has been all about *undirected* graphs. However, some graphs are *directed* graphs, in which each edge is directional, that is, the edge (u, v) is not the same as the edge (v, u). Create an abstract subclass of `Graph` called `DirectedGraph`, and write any necessary code to make it represent and compute with directed graphs. Modify your implementation of one of `EdgelistGraph`, `MatrixGraph`, or `DictGraph` to implement the `DirectedGraph` abstract class.

Submission

To submit this assignment, upload a **`.zip`** file containing both Python files to the "Homework 6" assignment on Carmen. As always, be sure to note all group members who contributed to the assignment and what those contributions were.