Module 11 Homework - Graphs

Contact Tracing

Graphs can be used to model connections between people. In order to determine whether a person is at risk after exposure to a pathogen, through contact with other people, we are going to design classes and methods that can be used for contact tracing.

Certain pathogens can infect people through mutual contacts, such as airborne pathogens. Two people talking could potentially contaminate each other. Other pathogens contaminate in a unilateral way, such as bloodborne pathogens. A blood donor could potentially contaminate a recipient, but not the other way around. In our model, these two cases will be modeled by undirected and directed graphs respectively.

Unless otherwise specified, the order of the people in the graph is not important. The edges in the graph are unweighted.

Design and initialize the graph classes

The class `ContactGraph` implements a directed graph. The class `MutualContactGraph` inherits `ContactGraph` and implements an undirected graph. Both classes support the same methods and have the same functionalities, even though the outputs might be different. Make sure not to reimplement more methods than necessary in the child class.

- The graphs should be initialized using a collection of vertices and a collection of edges, represented by tuples

```
>>> V = [1, 2, 3]
>>> E = [(1, 2), (2, 3)]
>>> G1 = ContactGraph(V, E)        # Two directed edges:   1 ==> 2 ==> 3
>>> G2 = MutualContactGraph(V, E)  # Two undirected edges: 1 --- 2 --- 3
```

Deliverable

Write the class definitions and the initializers of the `ContactGraph` and the `MutualContactGraph` classes.

Make the classes iterable

The classes should be iterable. The iteration is to be done over the nodes of the graphs.

```
>>> for v in G1: print(v, end=' ')
1 2 3
```

Deliverable

Implement the magic method

- `__iter__(self)`

Access the neighbors of a node

The method `neighbors(v)` should return an iterator over the neighbors of the node `v`

```
>>> for n in G1.neighbors(2): print(n, end=' ')
3
>>> for n in G2.neighbors(2): print(n, end=' ')
1 3
```

Deliverable

Implement the method

- `neighbor(self, v)`

  `v` is a node in the graph

Access all contacts of a node

The method `all_contacts(v)` should return an iterator over all the nodes reachable from `v`, including the node `v` itself. The nodes should be ordered by non-decreasing distance from `v`.

```
>>> for c in G1.all_contacts(2): print(c, end=' ')
2 3
>>> for c in G2.all_contacts(2): print(c, end=' ')
2 1 3
```

Deliverable

Implement the method

- `all_contacts(self, v)`

  `v` is a node in the graph

Access all close contacts of a group of nodes

The method `group_contacts(V)` should return an iterator over all the neighbors of the nodes in the collection `V`, not including the nodes in `V` themselves. A node that is the neighbor of more than one of the nodes in `V` must *not* be used more than once.

```
>>> for c in G1.group_contacts([1, 3]): print(c, end=' ')
2
>>> for c in G2.group_contacts([2, 3]): print(c, end=' ')
1
```

Deliverable

Implement the method

- `group_contacts(self, V)`

  `V` is a collection of nodes in the graph

Access all nodes closer than a given threshold

In an unweighted graph, the distance between two nodes is given by the smallest number of edges on any path between them. The method `contacts(V, d)` should return an iterator over all the nodes at a distance no more than `d` from the node `v`. The node `v` should be included. The iterator should return a tuple consisting of the node and its distance from `v`.

```
>>> for c in G1.contacts(1, 2): print(c, end=' ')
(1, 0) (2, 1) (3, 2)
>>> for c in G2.contacts(1, 1): print(c, end=' ')
(1, 0) (2, 1)
```

Deliverable

Implement the method

- `contacts(self, v, d)`

  `v` is a node in the graph, `d` is a distance

Submission

At a minimum, submit the following file:

- `contact_graph.py`

Students must submit to Mimir **individually** by the due date (typically, the second Wednesday after this module opens at 11:59 pm EST) to receive credit.

Grading

- 10 - class definition
  - 10 - inheritance used correctly
- 10 - `iterator`
  - 5 - the classes are iterable

    - 5 - functionality
- 20 - `neighbors`
    - 5 - iterator
    - 15 - functionality
- 20 - `all_contacts`
    - 5 - iterator
    - 15 - functionality
- 20 - `group_contacts`
    - 5 - iterator
    - 15 - functionality
- 20 - `contacts`
    - 5 - iterator
    - 15 - functionality

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on several assignments