CMPUT 275 Wi18 - INTRO TO TANGIBLE COMPUT II Combined LBL Wi18

Exercise 2: Coding Portion

This coding portion has three main parts:

1. Implement a function that efficiently counts the number of connected components in an undirected graph (modelled as a directed graph where both (u,v) and (v,u) directed edges are present for each undirected edge uv). (6 marks)

2. Implement a function that reads the data in a *comma-separated values* (CSV) file that describes a city's road network and returns an instance of the Graph class developed in the lecture (and included with the files for Assignment 1) corresponding to the **undirected** version of that road network.

   That is, for every edge in the graph file you should add both directions of that edge to the directed graph. (3 marks)

3. When your code run via
   `python3 count_components.py`
   it should build the undirected version of the Edmonton road network, call the function to count the number of components, and then print this number to the terminal. (1 mark)

You will add the above-mentioned two functions to a new script **count_components.py**.

The script should work when placed in the same directory as the files `graph.py` and `breadth_first_search.py` in the Google Drive directory for lecture 5 as well as the .txt file with the Edmonton graph from Assignment 1. All your code should be contained in the file `count_components.py`. Do not modify the script files we provided.

Submit only `count_components.py` and do not zip it. If you needed to modify the other files to get a working solution, then submit them all in one .zip and include a README file describing the changes. However, modifying the other files will result in a deduction.

The details of the individual tasks mentioned are described below.

1) Counting Components

Implement a function **count_components(g)** that takes a single parameter **g**, an instance of the class **Graph** that we constructed in the lectures. The function should return a single integer that is the number of connected components in the graph **g**. This function assumes for each directed edge (u,v) in **g** that there is also a directed edge (v,u) in **g**. As discussed in the lectures, this is a way to model an undirected graph using our directed graph class.

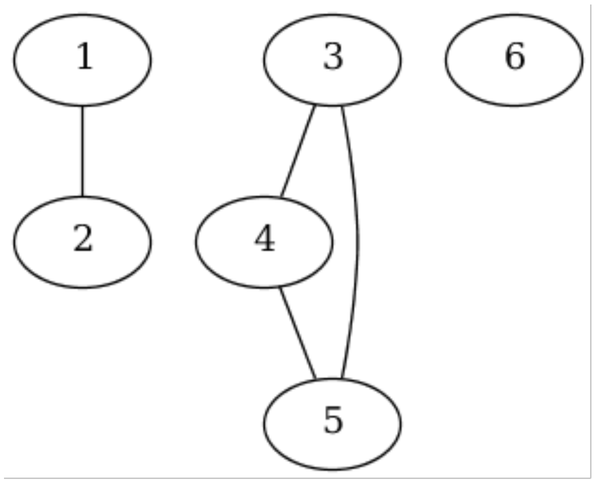
Recall a connected component of an undirected graph is a subset of vertices C such that there is a path between any two nodes in C and there is no path between a vertex in C and a vertex lying outside of C.

For full marks **this function should run in linear time**, i.e. O(*n*+*m*) where *n* is the number of vertices in the graph and *m* is the number of edges. Recall that our convention is to say that a single dictionary insertion or lookup take O(1) time.

An example of an interaction is the following. It uses ipython3, which is started from the same directory where all the scripts are.

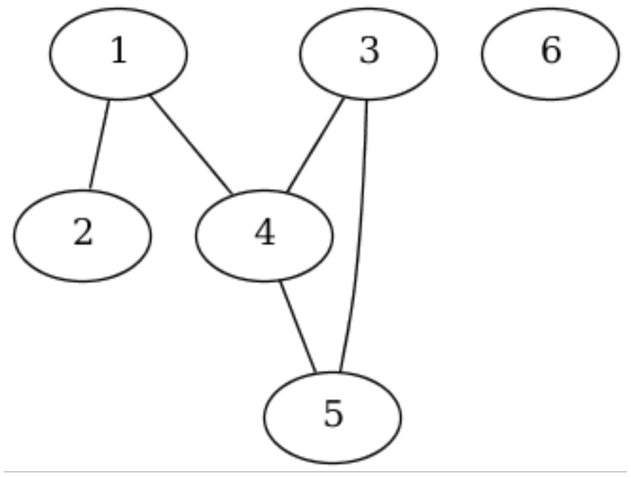
```
>>> from graph import Graph
>>> from count_components import count_components
>>> g = Graph()
>>> vertices = {1, 2, 3, 4, 5, 6}
>>> edges = [(1, 2), (3, 4), (3, 5), (4, 5)]
>>> for v in vertices: g.add_vertex(v)
>>> for e in edges: g.add_edge( (e[0], e[1]) )
>>> for e in edges: g.add_edge( (e[1], e[0]) )
>>> count_components(g)
3
>>> g.add_edge( (1, 4) )
>>> g.add_edge( (4, 1) )
>>> count_components(g)
2
```

To illustrate this interaction, the graph g in the first call to `count_components` looks like



and has connected components {1,2}, {3,4,5} and {6}. The function should return 3.

After the edge (1,4) is added, the graph then looks like



and the two connected components are {1,2,3,4,5} and {6}. The function should return 2.

2) Reading a Graph

Implement a function **read_city_graph_undirected(filename)** that takes the name of a plain text file as the argument. The file describes a road network. The function should return the graph (an instance of our Graph() class) corresponding to the undirected version of that graph.

The file format will be in exactly the same format as the Edmonton graph file. That is, there are two types of lines:

- A line of the form *V*,*ID*,*Lat*,*Lon* describing a vertex.

  Here *V* is the single character 'V', *ID* is an integer that is the vertex identifier, and both *Lat* and *Lon* are floating point numbers that describe the geographic coordinates the the vertex.

  Important: No two lines describing a vertex will have the same *ID* value. The vertices of the graph you return should be the *ID* values of these lines.

- A line of the form *E*,*start*,*end*,*name* describing an edge/street.

  Here *E* is the single character 'E', *start* and *end* are the IDs of two vertices connected by the edge, and *name* is a nonempty string giving the name of the street.

  Important: There may be spaces in *name*, but no commas. Every vertex ID in an edge will appear earlier in the file in a vertex description line.

The following is an example of a very simple city with 4 vertices and 3 edges.

```
V,1,53.430996,-113.491331
V,2,53.434340,-113.490152
V,3,53.414340,-113.470152
V,4,53.435320,-113.480152
E,1,3,St Albert Road
E,2,3,80 Avenue  North-west
E,2,1,None
```

The file describes a directed road network, but you should treat each edge as undirected in this exercise. Note that the geographic coordinates and street names are not used in this part (the coordinates will be used in assignment 1).

3) Components in the Edmonton graph

Your last task is to simply count the number of components described in the undirected version of the Edmonton graph. The file is found with the files for Assignment 1 on eClass. You should print this number when the script is run via

```
python3 count_components.py
```

However, you should not just hard code the integer and print it. Your script first should build the undirected version of the Edmonton graph using `read_city_graph_undirected("edmonton-roads-2.0.1.txt")`
You must call the function with exactly this file name in your code (the file name should be hard-coded into your script). Then your script should print the value returned from the `count_components` function when it is supplied with this graph.

Why will there be more than one? Because we obtained this data from OpenStreetMap by asking for all vertices and edges in a bounding box around Edmonton. This omitted the roads that crossed the boundary of our box, thus disconnecting some small parts near the border.

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Graded
Due date	Monday, 29 January 2018, 11:55 PM
Time remaining	Assignment was submitted 3 hours 25 mins early
Last modified	Monday, 29 January 2018, 8:29 PM
File submissions	count_components.py Export to portfolio
Submission comments	Comments (0)

Feedback

Grade	10.00 / 10.00
Graded on	Monday, 19 February 2018, 11:55 AM
Graded by	Jesse Farebrother
Feedback comments	Not sure why you didn't want to pop from the `vertices` set. It is more efficient, behind the scenes when you do `for x in something` Python creates an iterator which will have to iterate over the entire set. If the set is large this could take a while.