```
Lab: Graphs (Part A)

Updated 1:15pm Sun March 4
   * fixed the traversal algo

Updated: 4:30pm Sat March 3
   * removed the requirement to return a tree
   * added path function

Due: in your lab session; week of March 4
Language: Python

A. G=(V,E)
   Let V be the set v0,v1,v2,...
   (i.e., with ascending non-negative indices)
   and E be the set e0,e1,e2,...
   (i.e., with ascending non-negative indices)

Create a class "graph", with the following methods:
* __init__(self)
  create an empty store for the graph, which will be an adjacency list

* addVertex(self,n)
  this will add "n" vertices to the graph, and return the value of the
  final number of vertices in the graph; the function may be called
  multiple times to add more nodes to the graph.

  The first time this is called (arg=1), it should return 1 and expand
  the store for the adjacency list to have one slot (the index 0 slot);
  The second time this is called (arg=1), it will return 2, and expand the store
  for the adj list to have two slots. Etc.
  It can also be called with any non-zero positive integer arg.

  If there is an error return -1.

* addEdge(self,from_idx,to_idx,directed,weight)
  where from_idx and to_idx are nonnegative integers
  and directed is either True or False
  and weight is any integer other than 0

  This adds an edge (a directed edge if directed==True, otherwise
  an undirected edge) from vertex<from_idx> to vertex<to_idx>
  with non-zero integer weight, weight.

  If there is an error return False, else True

* traverse(self,start,typeBreadth)
  These functions will return a list obtained from a breadth or depth
  traversal of the graph (based on typeBreadthFirst).

  If there is an error: return an empty list.

  start is either None or a non-negative integer:
  * if start==None: then the traversal must traverse the entire graph
    (i.e., including all of the subgraphs that may be disconnected from
    one another)
  * if start is an integer up to the maximum index of graph vertices
    then the traversal is just to whatever vertices that are connected
    to it (i.e., for which a path exists from the vertex with an index of
    start).
    If an invalid start index is entered, this is an error (v.s.).

  typeBreadth: True for Breadth; False for Depth
```

The basic algo for graph traversal is as follows, with
breadth traversal accomplished via C being a Queue, and
depth   traversal accomplished via C being a Stack

```
traverse(G=(V,E)):
   initialize C to empty
   initialize Discovered to have as many slots as there are v in V
   initialize Processed to have as many slots as there are v in V
   set all slots of Discovered to be False
   set all slots of Processed to be False
   for v in V:
      if Discovered[v] == False:
         store v into C
         Discovered[v]=True
      while C is not empty:
         retrieve w from C
         if Processed[w]==False:
            process(w)
            Processed[w]=True
         for x = all vertices adjacent to w
            if Discovered[x] == False:
                store x into C
                Discovered[x]=True
```

This algo will have to be slightly modified to handle the "start" index. Which line of code needs to be adjusted?

Return value:
a list consisting of all nodes visited via the traversal

* if start is set (i.e., is a valid integer) then this will be ONE list
* if start is not set, then this will be a list of lists (each sublist corresponding to a different connected subset of the graph)
  e.g., [[sublistA], [sublistB], [sublistC], ..., [sublistN]]
  if there are connected subgraphs A through N
  e.g., [[sublistA]]
  if the entire graph is connected

* connectivity(self,vx,vy)
  This returns a 2-list.

  Element[0] is True if there's a path from vx to vy, else False
  Element[1] is True if there's a path from vy to vx, else False

* path(self,vx,vy)
  This returns a 2-list.

  Element[0] is a list of vertices from vx to vy, if there is a path,
             otherwise []
  Element[1] is a list of vertices from vy to vx, if there is a path,
             otherwise []

  Include endpoints

Submit this with the standard submit command and an arg of 5
The file should be called graph.py. NO additional helper files should be used.