Project 8: Graphs

Due: Thursday, April 25th @ 8:00pm
This is not a team project, do not copy someone else's work.

Turning It In
Be sure to submit your project as a folder named "Project8" and include in the folder:

- Graph.py, a Python3 file
- ReadMe.txt, a text file that includes:
  - Your name
  - Feedback on the project
  - How long it took to complete
  - A list of any external resources that you used, especially websites (make sure to include the URLs) and which function you used this information for. Please note that you can not use outside resources to solve the entire project, or use outside sources for multiple functions that leads to the solution of the project, then you are submitting someone else's work not yours. Outside sources use should not be used for more than one function, and it should not be more than few lines of code to solve that function. Please note that there is no penalty for using the programming codes shared by Onsay in Lectures and posted on D2L

Assignment Overview

This will be an implementation of a Directed Unweighted Graph. This graph is intended to work for a stream of edges getting passed in, meaning this will not be a fixed size graph and the allocation could grow on each insert of edge pairing.

Generate_edges(size, connectedness):

Do not edit

- Method for creating a directed acyclic graph
- Returns a generator object that will yield strings that are space separated representing edges

GraphError Class

Do not edit.

- Inherited from Exception and should be raised when iterable is invalid during graph construction

Vertex Class

__init__(self, ID, index):
Do not edit.

- self.ID represents the unique identifier of vertex instance
- self.index represents which row in the adjacency matrix the vertex holds its outgoing edges in.
- self.visited represents a Boolean that acts a flag to determine when a vertex object has already been seen; starts as false.

in_degree(self, adj_matrix):

- Given an adj_matrix, type: List[List[Optional[Vertex.ID]], return the number of incoming edges to its vertex
- Return type: int
- adj_matrix should not be modified in this method
- Time: O(V) Space: O(1)

out_degree(self, adj_matrix):

- Same as in_degree, but return the number of outgoing edges to its vertex.
- Time: O(V) Space: O(1)

visit(self):

- Set the visit flag to seen (true).
- Return type: None
- Time: O(1) Space: O(1)

Graph Class

__init__(self, iterable=None):
Do not edit.

- self.id_map: map where the key is Vertex.ID and the value is the Vertex object
- self.matrix: graph representation of type List[List[Optional[Vertex.ID]]]
  - For each edge, the vertex ID of the destination vertex is stored in the appropriate indices.
  - For each edge, the row number is the source index and columns number is the destination index.
- self.iterable: some iterable object holding strings that are space separated expressing edges to be inserted into graph

__eq__ The equality operator is defined and shouldn't be touched and is used to validate and ensure the structure of the graph.

get_vertex(self, ID):

- Given an ID that is the same type as Vertex.ID, get the corresponding Vertex object.
- Return type: Vertex
  - If ID doesn't exist return None
- Time: O(1) Space: O(1)

get_edges(self, ID):

- Given an ID with type: Vertex.ID, return the set of outgoing vertex ID's from the input vertex
- Return type: Set[Vertex.ID]
- Time: O(V) Space: O(V)

construct_graph(self):

- Iterates through iterable and calls insert_edge to create the graph representation stored in self.matrix.
- Return None
- Time: O(V^2) Space: O(V^2)

insert_edge(self, source, destination):

- Creates vertex objects, if needed, then adds edge representation into the matrix between the given source and destination, and updates self.id_map
  - source type: Vertex.ID
  - destination type: Vertex.ID
- Return None
- Time: O(V) Space: O(V)

bfs(self, start, target, path=None):

- Does a breadth first search to generate a path from start to target visiting a node only once.
  - If no such path exists return an empty list.
  - start type: Vertex.ID
  - target type: Vertex.ID
  - path type: Optional[List[Vertex.ID]]
- Both bfs and dfs (below) are provided an optional additional parameter 'path'. This is to support keeping state in where you are in a recursive call stack, if you choose to implement either recursively.

- bfs and dfs (below) can return **any** accurate path from source to target.
- Must not access self.adj_matrix
- Return type: List[Vertex.ID]
- Time: O(V^2) Space: O(V)

dfs(self, start, target, path=None):

- Same as bfs, but doing a depth first search instead
- Time: O(V^2) Space: O(V)

Application Problem

Given a starting ID and value K, return all vertex ID's that are K (inclusive) vertices away.

```
ex.     ID = 100, K = 2
         Graph is
           100 -> 99
           99  -> 98
           98 -> 97
           99  -> 20
        result = {98, 20}
```

find_k_away(K, iterable, start):

- Parameters
  - K type : int
  - iterable type: some iterable holding strings that represent edges
  - start type : Vertex.ID
- Return type: Set[VertexID]
- K is guaranteed to be >= 0
- Should not access Graph.adj_matrix, or Graph.id_map explicitly.
- Time: O(V^2log(V)) Space: O(V)

Notes

- Unless specified in the function/method descriptions, you are allowed to access all class attributes anywhere in this project.
- You have full usability for list, set, dictionary methods anywhere in the project.
  - All other container classes must have prior approval on Piazza.
- When parsing through the iterable object, it will always hold strings, and the space separated string should be casted to integers.

- For time/space complexities notation, V denotes the number of vertices in the graph
- Visiting a vertex is defined, in this project, as when all paths have been explored starting from that vertex.
    - Think of it as "now done visiting"
- For all traversals, in the event you can traverse different edges at a single point, (and the result makes a difference), visit in ascending order of Vertex.ID

Rubric:

- Tests - 80
- Complexities - 20
    - BFS & DFS time and space __ / 10
        - here get_edges should factor in implicitly.
    - insert_edge/degrees/construct_graph time and space __ / 5
    - Find_K_Away time and space __ / 5

Project was Authored by Cyndy Ishida and Yash Vesikar
ACTIONS