CS310 Algorithms

Project 2: Graph Project

Submitted: December 11th, 2017

Group Members and Responsibilities:

Since this project asked us to implement functions and ADTs that are deeply interconnected, we had to work together for most of the parts, since it was difficult to continue off on someone else work. For the purpose of this project, we divided the work as follows:

1. Abhinav R. Pandey & Giorgi Kharshiladze: Worked together in order to implement DFS, connected components and the cyclic test. They also implemented the graphBuilder function in order to read files, and the showGraph function to test the files. Abhinav wrote the project report, and Giorgi tested our program and developed the typescripts.

2. Sandro Alavidize & Karan Shrestha: Implemented the Graph Abstract Data Type, including the subclass Node and the subclass Edge. Sandro and Karan also implemented firstEdge, addEdge, searchCity, and additional selector/mutator functions for the ADT.

Design of the Project:

Our project implements a graph abstract data type in Python, and uses this ADT to construct a graph from a given input file. The graph ADT consists of a parent class 'Graph' with its associated graph functions, and two sub-classes 'Node' and 'Edge'. A variety of functions that manipulate the graph and return the information stored in the graph have been implemented. The graph is implemented as an adjacency list, where each of the nodes is stored in a list and each node is associated with an adjacency list consisting of all of its edges. Our graph ADT is only applicable to undirected graphs.

Using the Graph ADT, we have also a function that constructs a graph from a given input file. Once the graph has been constructed, we are able to print the graph in the form of the test graph (graphSmall1.in) given in the project description using showGraph. This allows us to ensure that our Graph ADT is constructing the graph from the input file properly. We have also implemented a function DFS that conducts a DFS traversal of the graph. The DFS function continues until all of the nodes in a single connected component have been traversed, then returns a list of all of the visited nodes. Using this list, we identify the connected components by removing the elements present in the visited list from the node list of the graph. We then continue to call the DFS function for the unvisited nodes, until all of the nodes have been visited. Connected components are stored as a dictionary.

Additionally, we have also implemented a cyclic test to determine whether or not the graph contains a cycle. The function checkCyclic checks if a cycle is present in the graph by utilizing a stack and a visited list. The stack is used to backtrack during the traversal. If an element occurs twice in the visited list after traversal using the stack to backtrack, then the graph contains a cycle.

Future Work

Since our implementation of the Graph ADT is only for an undirected graph, we could implement an ADT that includes directed graphs as well. We could also implement our Graph ADT with other information besides cities, and add additional functions that can manipulate and retrieve information stored in the Graph ADT.