Documentation – SocialSim

Overview

The main parts of the program are SoicalSim.java and FileReader.java, the rest of the classes are either data structures or just used to store an Object that I needed for the program such as Person.java which stores information about a person. Currently the network and event file are read in and created by the FileReader class which then calls methods in SocialSim to simulate the spread of information in the network. The readEvent file currently reads in each event line by line and immediately runs the event inside the same method, if given more time this would be changed to store the events in queue which the method would return. SocialSim would then use this queue to execute each event one at a time writing a timestep after each event. Meaning one timestep would be defined as the time taken to execute one event.

SocialSim – Used to run the program and contains the majority of methods used for interactive mode and simulation mode. Also contains the menu/user interface and user input methods which, in hindsight, should have been contained in a separate menu class to make SocialSim shorter in length and only contain the important methods used for the propagation of information. Uses IllegalArgumentExceptions to handle user input errors.

FileReader – Class that contains all file io methods. Created to keep these methods separate from the rest of the program and group them together. The methods are used to read in networks and events from a file, write timesteps to a file and save and load DSAGraph Objects.

DSAGraph – Used to create and store a custom graph and perform all necessary actions on that graph such as adding a new vertex. Used by SocialSim and FileReader to store the social network as a graph and alter different parts of the graph.

DSAGraphNode – Class to store information for each vertex/node in the graph and contains methods to adjust this information. In this program each graph node stores a Person object representing each person that is in the social network, allowing the graph to keep track of all people and their data. Also stores the name of that person as a label so that the graph can find certain nodes without having to access the Person object. Stores a DSALinkedList called links which contains other graph nodes that represents followers of the person the graph node stores.

DSAEdge – Class to represent relations between each node/person. Stores a start node and an end node showing the direction of the relationship. If node ‘a’ is the start node and node ‘b’ is the end node, then ‘a’ follows/relates to ‘b’. Also stores a boolean that determines whether the edge is directed or not (two-way relationship vs one way). Stores a unique label sued to identify the edge. Label is a combination of the start node’s label and the end node’s label. The class also houses all methods required for setting and getting these values.

Person – Class used to store information for each person in the social network. Created to contain this information instead of just storing its state ass class fields in a graph node. Stores a linked list of Post objects which represent any posts that the person has made. Contains accessors and mutators for these values. Also originally contained lists to keep track of followers and who the person was following however, this seemed like a waste of space as there is other ways to get this information in the program.

Post – Stores the state of each post made by users of the social network. Could have been created as a private inner class in Person, but I chose not to as to keep the code neater and as this class is used by SocialSim and FileReader. Posts’s class fields are: name - a String to store the name of the person who posted the post (also could have stored a reference to the Person object itself), caption – a String containing the text that a person is trying to share, likes – an integer representing the number of likes on the post. Contains accessors and mutators for this information and a method to display this.

DSALinkedList – Is a linked list data structure, used to store a list of objects to keep this data together. Class fields: head - a DSAListNode that points to the first node in the list, tail – a DSAListNode that points to the last node in the list. These nodes contain the information that need to be stored in the list. This class is used by SocialSim, Person, DSAGraph and DSAGraphNode so it is a very common data structure used throughout the program. This is due to its dynamic size as the list only gets full if you run out of memory, as this program can be used to add Person, DSAGraphNode, DSAEdge and Post objects to lists, the count of the data stored is always changing. Houses methods for finding inserting and deleting values in the list and also for iterating through the list.

DSALinkedListIterator – Private inner class of DSALinkedList created to allow users to iterate through the list but only view the values stored in the list and not the actual list nodes themselves. Stores a DSAListNode that points to the current node in the list that the user can perform actions on.

DSAListNode – Used by DSALinkedList to hold information about an object that the user is trying to store and also keep track of adjacent list nodes so that we can iterate through the list to perform actions such as finding, inserting and removing nodes from the list.

DSACircularQueue – Container class to store a queue. Class fields: queue[] – an array of Objects that are stored in the queue, head – an integer representing the index of the front item in the queue, tail – an integer that stores the index of the last item in the queue. Contains methods to add, remove and view items that are stored in the queue. Used by DSAGraph to store graph nodes while performing a breadth or depth first search. Also used by SocialSim and FileReader to store how many people in the social network that a post has reached.

Justification

In the propagate method in SocialSim I use two DSACircularQueue’s to store DSAGraphNode’s when searching through the graph. One of these stores all the graph nodes/people who have liked a post so that a Post can be spread to these people’s followers and the other stores every node that the post reaches, representing every person that has seen the post regardless of whether they have liked it or not. I have used queues as this algorithm is a modification of a breadth first search and they are a first in first out (FIFO) data structure which is important as the algorithm spreads the information starting from the node who made a post and moving outwards to all their followers. This ensures that the spread of information will be in order of followers.

Each Post is stored as a DSALinkedList inside the class Person. I used a linked list to store these as a person can make as many posts as they want so the posts storage needed to be dynamic. I could have stored them in an array of posts, however arrays are a fixed size and if the array got full then I would have to transfer all elements to a new larger array. On a small scale this is fine, but if the person had a large number of post’s this would take a fair bit of work to move every element. Although this linked list takes O(N) to traverse the list and reach places to insert and delete, its’ dynamic structure was the main reason I chose this ADT. DSAGraphNode also uses a linked list to store the adjacent nodes to itself (nodes/people that are following it). I used a linked list here for the same reasons as above, a person’s followers are constantly changing as people can follow and unfollow, so it is better for the size to be dynamic. The issue with choosing these linked lists is the overheads of memory as the list is double-ended and double-linked meaning each list node stores a pointer for the next node, the previous node and to the data the node is storing. On top of this there is also a pointer that points to the last node in the list. This is okay for small networks, however for much larger networks on a bigger scale this may become a problem and be an unnecessary usage of memory. DSAGraph.java and DSAGraphNode.java were previously submitted for practical 5 and have since been modified. DSAListNode.java and DSALinkedList.java were previously submitted for practical 3. DSACircularQueue.java and DSAQueue.java were previously submitted for practical 2.

