Social Network Analysis Report

Abstract

Two object classes are made to represent a person in the network and a post to represent information which can be spread throughout the network. A person class contains the information of the person's name, their followers, who they're following and the posts they made. These information are stored in a linked list to keep track of information the person has without worrying about the size limit. The Network.java class uses a HashTable and a LinkedList to keep track of the people in the Network and their posts. The combination of these two data structures results in a time complexity of (or close to) O(1) to get a Person object from the Network by name, rather than having to iterate over the linked list to verify the matching person, which was the original data structure used in the previous Graph class. Using a LinkedList to store Post objects results in faster search time for Post objects rather than having to iterate through all active Users in the Network as they may not have made a Post.

Alterations and additions to data structure codes were made in order to suit the specifications, ie. Set() method was implemented in the HashTable class to iterate over the table, returning non-null/active values stored in a LinkedList and the remove(Object) method for efficient removal of edges/followers for Vertices .

The old graph prac had a Vertex class which contained a boolean to mark the vertex as visited which is used for graph traversal. With modifications in an improvement of the old Graph, a HashTable implementation of the Network had more benefits over a LinkedList, such as key value mapping, retrieval/removal/adding Nodes, in this case People in the Network class are mapped out by their String name which makes Node and Edge operations as close to O(1).

Sorting the data felt unnecessary as the Network class implementation used a HashTable class. Sorting out the posts in the linked list also was not needed as storing the post objects in the linked list to a Priority Queue was satisfactory to perform listing users/posts in the graph from highest to lowest. The time complexity overall cost O(nlog(n)+nlog(n)) . It is essential to iterate over each value in linked list to store each value in the max heap which is n_items*log(n_items) as the time complexity of storing items in a max heap cost O(log(n)). the extra nlog(n) comes to removal for all the vertices stored in the max heap to be displayed to the user. If a search algorithm was used, the time complexity of list functions would be O(n^2) for basic sorts, though using a linked list to store Object values a better approach would be merge sorting the linked list, resulting a sorted data in O(nlog(n)) time. This idea of sorting felt too complicated to implement at the time as it requires sorting a linked list of Object values so it was difficult to a linked list of objects without typecasting it back into an object, this also felt inconvenient because the linkedlist was planned to store any data type into an object. If a sorting algorithm was implemented, it would require O(k+n+nlog(n)) for this implementation, O(k) to go over the HashTable array and store each valid object in a linked list, O(n) for displaying the list from highest to lowest and O(nlog(n)) for the merge sorting.

HashTable Vertices

The time complexity of a HashTable is (or close to) O(1) for retrieving/removing values in a fixed size array. Because the Network stores Vertices represented by People and their name, a HashTable allows mapping/storing Object values by key values which in this case is a String name. The program, does not allow to store duplicate key values. The major trade-offs to using a HashTable is the space used as a fixed size array is created, in an occurrence of a low Network population, space is wasted as memory was allocated for the HashTable array. With the sacrifice of space, the HashTable class allows fast and speed efficiency in Node and Edge operations, as it only needs to map out the vertices given by their key value as close to O(1). The addition of Set() function was implemented so that it solved the problem with Graph traversal, such as breadth-first search. Set() function returns the list of values that was stored in the HashTable.
The function checks every cell in the array for active/non-null objects and stored in the LinkedList, this function is my own implementation based off the keySet() from the built in Hashtable in java which returns a Set of key values in the Hashtable.

LinkedList Post

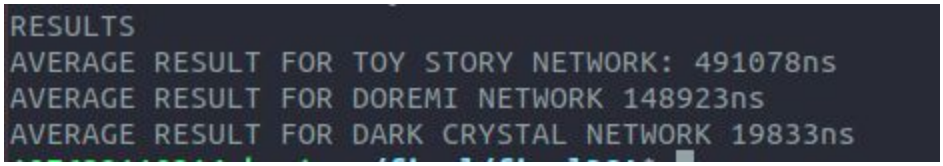
The idea of storing Posts in a LinkedList felt, the best solution at the time as having to iterate through all the Vertex People and checking if they made a post would come at the cost of speed as a Person Vertex may not have made a Post yet, so the Network class contains a “pool” of Posts created by People in the Social Network. Storing Posts using the HashTable class is a difficult task to do as many People can make multiple Posts that are the same and Posts can be the same and made by different People. A Private Inner Post class was implemented to store the value of the message, the Person identity and the LinkedList of people who liked the post.

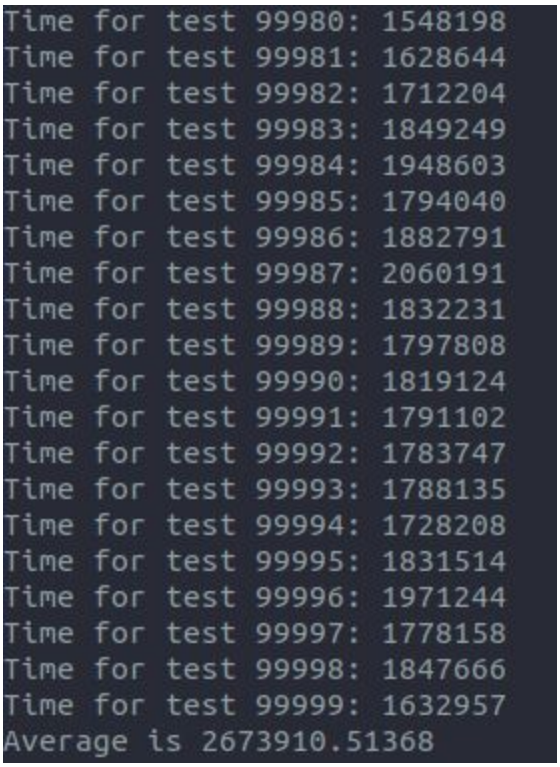
Time Step

breadth-first search was implemented to go through each vertex and their edges starting from the source of the post, the probability of liking and following is represented by 2 integer parameters, what triggers the breadth first search to occur in a Node is the probability of liking the post, if the probability occurs, the current Person Node visiting will like the post, then the followers are added into the Queue which will then have a chance to like the post and follow the original poster. The time step function is a nested for-each loop going through the posts and the followers of the original poster, calling breadth-first search for the total number of followers the original Poster has. A probability function was implemented using Java’s built in Random number generator class. The function imports an integer, the number generator function is called to generator a random number from 0-100 which works as: Let k = prob_like, n = random number generated. If n <= k then the function returns true, else returns false, this will work for the breadth-first search like/following. Two functions implemented, one importing the probability of liking and the other imports the clickbait factor and the probability of liking. The HashTable was very useful, as it was also a container for storing all visited Nodes during the Breadth-first search, meaning it was not necessary to stored a boolean or a marker for visited in each Person Object created which saves space.

Results

The simulation time step was tested for all three Network files. The results show that as the graph grows in size, the performance of the overall time step slows as there are more vertices to traverse through in the graph as well as calculating the probabilities and iterating over the posts. The network Do Re Mi had the fastest time, followed by the dark crystal network and toy story with the slowest performing graph time step.This was all due to the larger network connections between the toy story graph and the others where Do Re Mi, network vertices only had a few vertices in total where as the toy story network continued to increase in vertices as the event file was loaded and time step occurred.





Credit:
DSALinkedList files from DSA practicals
HashTable files from the DSA practicals
DSAQueue files from the DSA practicals
Graph giles from the DSA practicals
Data Structures and Algorithms in Java, Second Edition Robert Lafore implemented the code segment for trickle down algorithm