Annelise Thorn

Professor Alfonso Bastias

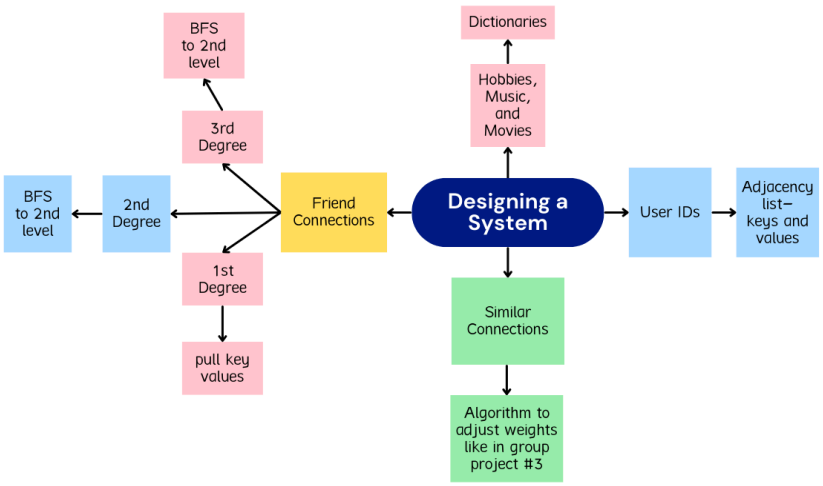
DTSC 5501: Data Structure and Algorithms

13 December 2024

Final Exam - Part B Report

Question 1: Designing a System

Mind Map



General Approach to Solution

For my solution, I created a social network style graph that represents connections between users and added a function that allows friends to see suggested friends with similar interests. The graph I used is an adjacency list, where each key is a user ID and the value is a dictionary of their friends. I also created separate dictionaries of users' hobbies, music, and movie preferences, so that I could create suggested connections based on similar interests.

Since first_degree_connections() shows immediate connections, and the adjacency list I created shows immediate connections, I created a function which simply pulls the keys from the adjacency list for a specific user_id. For second_degree_connections() and third_degree_connections(), I used breadth-first search with a queue to explore connections starting from the users' immediate friends or their friends' of friends. Each level of depth is tracked to ensure that the search stops at the desired level of connection.

Suggest_connections_based_on_intersts() looks at users' hobbies, music, and movie preferences to suggest new friends. It compares a user's interests to every other user's interest, and if they share any similarities, a suggestion is made and a new dictionary suggested_graph is built. In suggested_graph, each user has a list of potential friends that they are not connected to yet but share at least one common interest.

In summary, this code builds a social network graph and explores connections and users' interests. By using breadth first search to search the graph, it allows users to identify different degrees of connections and could be further explored beyond 2nd and 3rd degree connections. Additionally, more interests and more weights to said interests could be added to improve upon connection suggestions.

Explanation of the Data Structures

The data structures used in this solution include dictionaries, sets, queues, and a class. I used an adjacency list with an outer dictionary representing user IDs and an inner dictionary representing users' friends and a weight/identifier. I used an adjacency list because it's easy to add nodes and form new connections, which is the goal of social network sites. Adjacency lists are also easy to traverse and I knew this would be efficient for finding second and third degree connections. Additionally, I wanted a way to update the strength of the connection between individuals and I knew this would allow me to have weighted edges that I could modify based on mutual interest. I also used dictionaries to represent the user's hobbies, preferred music genres, and preferred movie genres, because these could be easily modified or added to. For second_degree_connections() and third_degree_connections() I used sets to manage unique user connections and visited users so I could ensure I wouldn't have any duplicates. I also used queues in these functions to perform breadth-first search so that I could process nodes level by level. Lastly, I used a class structure to organize the graphs and related methods so that they're easier to manage and modify.

Algorithms For Retrieving 1st Degree Connections

Since user's 1st degree connections are already listed in the adjacency list, I created an algorithm to take the user_id as input and retrieve the list of user IDs directly connected to the

specified user_id by extracting the keys of the inner dictionary corresponding to that user. If the user_id does not exist in the graph it returns an empty list.

Algorithms For Retrieving 2nd Degree Connections

To find users' 2nd degree connections, sets are initialized to store second_degree_connections and visited to keep track of the nodes already processed. A queue is created to start with a given user at a depth of 0. Second_degree_connections() then traverses the graph using breadth-first search and processes the queue until all relevant nodes are visited. If the current user is more than 2 levels away it skips further processing. It then marks the nodes as visited and returns the second degree connections.

Algorithms For Retrieving 3rd Degree Connections

Third_degree_connections() does the same thing as second_degree_connections(), but instead of making sure the current user is no more than 2 levels away, it makes sure the current user is no more than 3 levels away and returns the third degree connections.

Algorithm for Suggesting New Connections to Users

Suggest_connections_based_on_interests() iterates through users, skipping existing connections and calculates a similarity score (0-3) based on users' hobby, music, and movie preferences for each pair. The pair gets one point per category of shared interest. The function then suggests connections if they share at least one interest and shares the similarity score so pairs can see how similar they are.

Analysis of the Time and Space Complexities

First_degree_connections() has a time complexity of O(n) and a space complexity of O(n). Second_degree_connections() and third_degree_connections has a time complexity of O(k1+k2) where k1 is the number of first degree connections and k2 is the number of second degree connections and a space complexity of O(n+k2) where n accounts for the visited set and k2 is the second degree connections. Third_degree_connections() has a time complexity of O(k1+k2+k3) which is the same as second_degree_connections_but accounts for third degree

connections k3 and a space complexity of O(n+k3). Suggest_connections_based_on_interests() has a time complexity of O(n^2) and a space complexity of O(n^2).

In terms of scalability, these data structures and algorithms would translate well for a larger scale, however, I would make adjustments to suggest_connections_based_on_interests() so that only the top 3 most common friends come up as a suggestion. Due to limited time constraints, the algorithm currently shows all individuals with at least one point of common interest, which if users have a lot of friends or a lot of hobbies could result in a lot of suggested users. However, the data structures used are easily scalable for billions of users and can easily add and remove users' connections and interests. Additionally, you could combine the first degree, second degree, and third degree algorithms so it tells you all this information at once, which could be more efficient, however, I was not sure if this information needed to be presented at the same time or not.