Department of Computer Engineering, METU

CENG 140

Section 1 & 2

Spring 2018-2019

THE 2

Yusuf Mücahit Çetinkaya
yusufc@ceng.metu.edu.tr

Due date: 19.05.2019 Sunday, 23:59

Linkedlist

1 Overview

You are working for a large social network company, **Linked***list*. Your team leader wanted you to take care of the backend. You will implement the fundamental methods for the social network company.

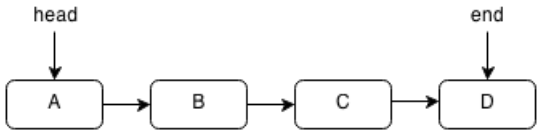
Linked*list* is a social network for professionals. It's like Facebook for your career. You might be surprised how useful this popular business social network is. Users are able to connect other users, show users' profiles and get the connection degree, get common connections with a user, etc. You are going to implement business logic as a backend developer.

Keywords: HashSet, LinkedList, search

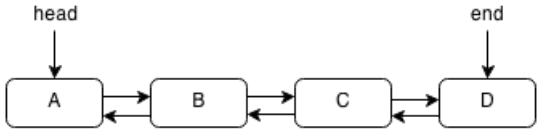
2 Background

2.1 Linked List

A linked list is an implementation of a queue that uses a chain of pointers. Instead of storing all the data in a fixed set of memory, we can store each element by itself but have a pointer from each element to the next element. A pointer is something that holds the memory location of another object.

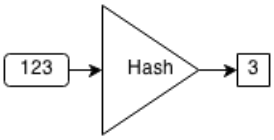


A linked list is similar to an array but it is different such that it is not stored in one block of data. Each element can be stored in a random place in memory but each element contains a pointer to the next element thus forming a chain of pointers. Think of a pointer as a clue to the next chest. Since the elements aren't in a block, accessing an element must be done by traversing the entire linked list by following each pointer to the next. However, this also allows insertion and deletion to be done more quickly. In a linked list the links only go forward and you cannot move backward. However, a doubly linked list is a linked list that has pointers going backwards as well as forwards.[2]

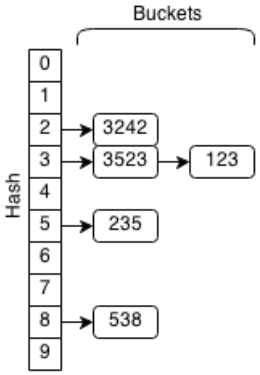


2.2 Hash Set

Hash sets are sets that use hashes to store elements. A hashing algorithm is an algorithm that takes an element and converts it to a chunk of a fixed size called a hash. For example, let our hashing algorithm be (x mod 10). So the hashes of 232, 217 and 19 are 2,7, and 9 respectively.



For every element in a hash set, the hash is computed and elements with the same hash are grouped together. This group of similar hashes is called a bucket and they are usually stored as linked lists. If we want to check if an element already exists within the set, we first compute the hash of the element and then search through the bucket associated with the hash to see if the element is contained.[1]



3 Tasks

You will implement some fundamental methods to manage user operations such as connect, get common connections, etc. An environment struct is used for storing the users. There is a user struct that contains the user's id, name, connections. Connections are stored as a hash set data structure. For making it easy, connection structure is given to you. The connection structure is a doubly linked list node that contains previous and next connection nodes. Additionally, the users of the connection and the connection time is stored in this struct. Input is retrieved from a file. Scanning the file and initializing the environment is done for you. Do not forget to work with pointers, which is essential to work always on the same objects!

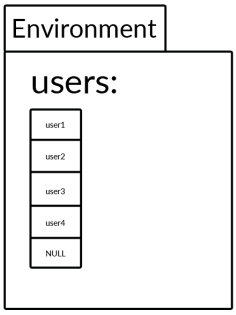


Fig1. Environment object that holds users. Last element is NULL.

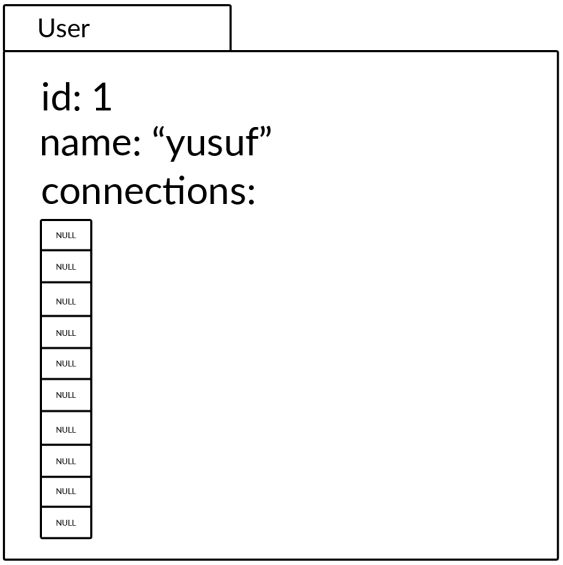


Fig2. An example of initial User object without any connection, with BUCKET SIZE of 10.

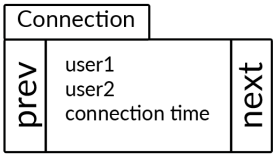


Fig3. Structure of Connection object.

There are some util methods that are implemented for you to use.

- **int letter_order(char c):** Returns the order of the letter in the alphabet.
- **void print_connections(User *user):** Prints the connections as stored in the hash set for debugging.
- **Environment *init_environment(char *user_file_name):** Initializes the environment with given input file.

The methods you are expected to implement are;

```
User *get_user(Environment environment, int id);
void connect_users(Environment environment, int id1, int id2, long connection_time);
Connection *get_connection(Environment environment, int id1, int id2);
void remove_connection(Environment environment, int id1, int id2);
User **get_common_connections(Environment environment, User *user1, User *user2);
```

3.1 Hash Code

This method is already implemented and given to you. It calculates the unsigned long hash of the given user. The user is sent via a pointer. The calculation is done with the name of the user and a hash key which is defined with a macro in **the2.h** header file. Starting from the first character of the name, the letter order is multiplied by power of 7 and the hash code is the sum of these multiplications. For example, if username is ”yusuf” and HASH KEY is 7, then hash values is computed as;

$$\text{”yusuf”} = \{25, 21, 19, 21, 6\}$$
$$hash = 25 \cdot 7^0 + 21 \cdot 7^1 + 19 \cdot 7^2 + 21 \cdot 7^3 + 6 \cdot 7^4$$

3.2 Get User

For this task, you are expected to find the User object stored in the environment by comparing id. This method takes an environment and id as a parameter and returns User* which is the address of the user object. You do not have the size of the dynamic users array. However, the last element is NULL.

3.3 Connect User

This method takes environment, id1, id2 and connection time as parameters. Connections are undirected, which is if user1 is connected to user2, user2 is also connected to user1. This connection information is stored in both users' connection hash tables.
Connections are stored in an appropriate field in the hash table according to their calculated hash. If the hash is bigger than the hash set size, modulo of the value is taken.

Let's assume the hash value of user1 is 22712 and user2 is 235263, connection time is 5 and HASH_SET_SIZE is 10. This connection should be located in the user1's hash set in the bucket with the index of 3 since 235263%10 is equal to 3 and in the user2's hash set in the bucket with the index of 2 since 22712%10 is equal to 2. If there is no other connection user1 and user2 have, the connections are stored in hash tables as shown below. Be careful, user orders are different for each user's connection record.

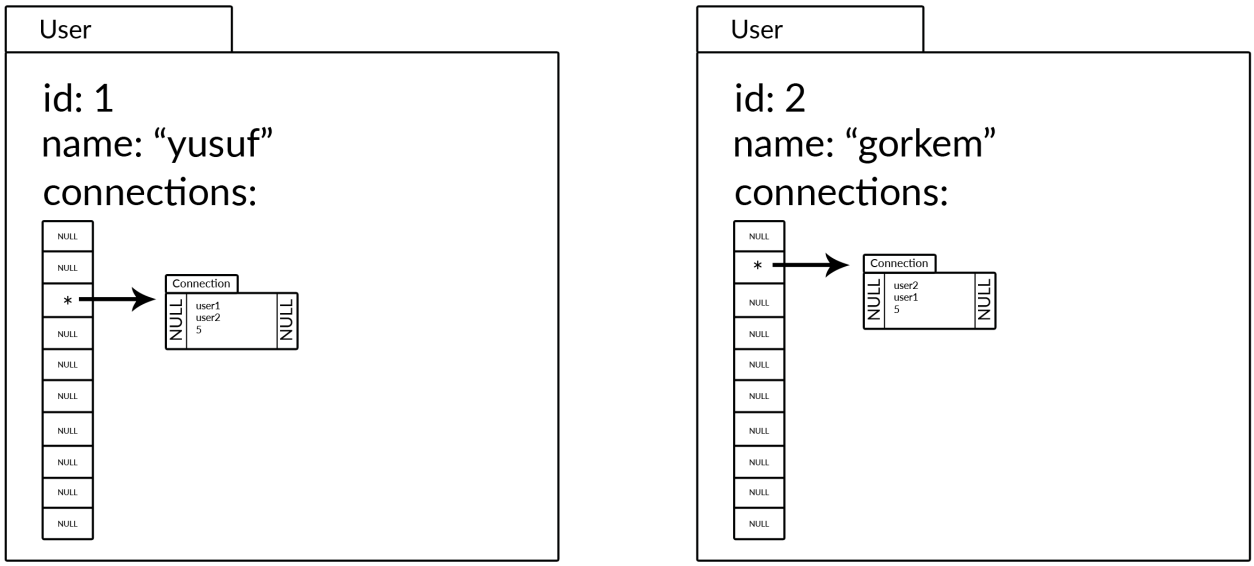


Fig4. How connections are stored in hash set for initial connections

The connection nodes in the bucket linked lists are sorted by their connection time in ascending order. If there are connections in the corresponding bucket, find the correct place and break the bonds between nodes and insert a new connection as shown below.

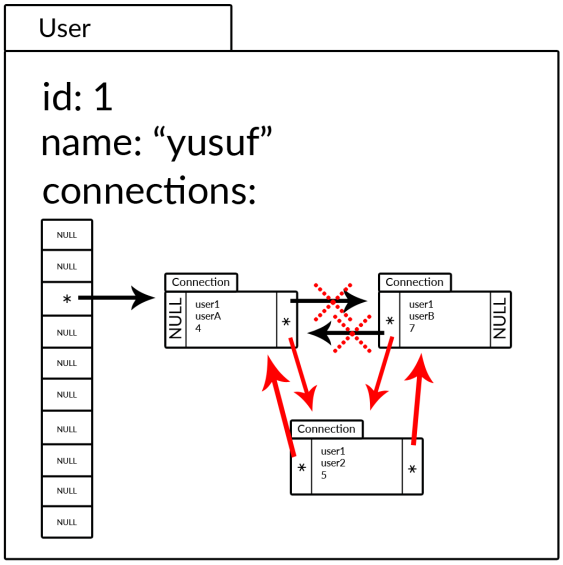


Fig5. How connections are inserted if there were connections before

3.4 Get Connection

For this task, you should find the corresponding Connection object stored in the user1's connection hash set. It simply takes environment, id1, and id2 as parameters. If there is no connection between user1 and user2 returns NULL.

3.5 Remove Connection

Similar to insert connection, it takes environment, id1, and id2 as parameters. Instead of inserting a new connection, it removes the existing connection. If there is no such connection, it simply ignores and does nothing. While breaking the bonds between nodes, you should connect the previous and next connection nodes to each other. A simple scenario is shown below. Additionally, do not forget to;

- Free the memory for the removed connection object,
- Remove the corresponding connection object from other user's connection hash table,
- Set the pointer in the corresponding bucket to NULL if there is no more connection node in that bucket.

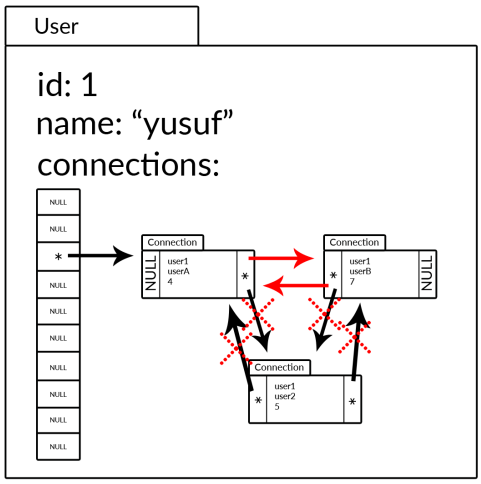


Fig6. How connections are removed

3.6 Get Common Connections

For this task, you will return the users that both of the users have a connection. It takes environment, user1, and user2 as parameters. The return type is a dynamic User* array. Similar to environment users, the last element is NULL. If there is no common connection, it returns a dynamic array with a size of 1 which contains NULL.

4 Regulations

- **Input-Output format:** You will not print anything as output. Tasks will be tested with separate main function like in the 3rd lab exam. The input file is for only initializing the environment and connections.

  Input Format:

  ```
  #Users#
  <user_id> <user_name>
  ...
  #Connections#
  <user1_id> <user2_id> <connection_time>
  ...
  ```

- **Programming Language:** C
- **Libraries and Language Elements:**
  You should not use any library other than *"stdio.h"*, *"math.h"* and *"string.h"*. You can define your own helper functions, provided that you have implemented 3 functions given to you in

Tasks section. Finally, please look at comments in the code for deep understanding about functions.

- **Compiling and running:**
  You should be able to compile your codes and run your program with following commands where run.c includes the main function:

  ```
  >_ gcc run.c the2.c -ansi -pedantic-errors -Wall -lm -o the2
  >_ ./the2
  ```

  Sample run.c file will be given to you.

- **Submission:**
  You will submit only **the2.c** file to the CengClass. Since submission is a part of the assignment, any error on submission file will be penalized. Late submission IS NOT allowed, it is not possible to extend the deadline and **please do not ask for any deadline extensions**.

- **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as an example. If your program gives correct outputs for all cases, you will get 100 points.

- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Sharing code between each other or using third party code is strictly forbidden. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.

References

[1] Michael Young. *Hash Set: The Computer Science Handbook*. URL: `https://www.thecshandbook.com/hash_set`. (retrieved: 03.05.2019).

[2] Michael Young. *Linked List: The Computer Science Handbook*. URL: `https://www.thecshandbook.com/Linked_List`. (retrieved: 03.05.2019).