CS 201 DATA STRUCTURES
ASSIGNMENT 4
Fall 2024

Instructions:

- *Late submissions will not be accepted.*
- *Please submit your code on Google Classroom; no email submissions will be accepted.*
- *Submit documented and well-written code in C++. Undocumented code will be assigned a zero.*

This assignment is an upgrade to the previous one. You will use the code you have developed in the last assignment and modify the Indices. Now, you will use the HASHING technique to develop Indices. All the changes that should be made are in the RED.

CUSTOMER SERVICE MODULE

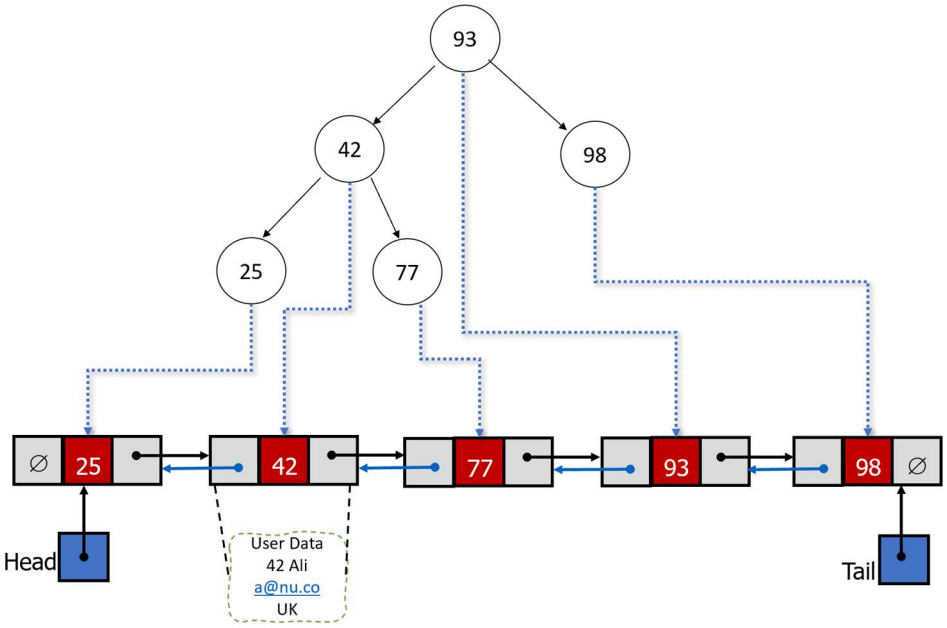
In this assignment, you are tasked with designing a **Customer Service Module** that tracks customers and the complaints they register on an online portal. This module will allow for effective complaint management, ensuring customer issues are recorded, tracked, and resolved **efficiently**. The design will include features to manage customer details and register complaints.

Users (Customers)

Maintain a complete list of registered users. Only registered users can submit a complaint. We record the following information about each user at the time of registration.

- UserID (a unique number assigned by our system to the users)
- User Name (a unique name selected by each user to represent him)
- Email (a unique email)
- Country (to track regions from which users are playing our games)
- Type (type of a user: platinum, gold, silver, regular, new)

The users' records are stored in a doubly linked list (DLL) sorted according to the UserID. To make searching efficient in doubly linked list (DLL) create an index using AVL tree on the attribute UserID. Each AVL tree node contains the UserID and the pointer to the User record in the DLL.



The figure above is an illustrative example of storing and organizing data. The user records are stored in a sorted, doubly linked list. An index is created on the UserID using an AVL tree that keeps the UserID and a pointer to the node of the doubly linked list that holds the user record. The dotted links are pointers to the nodes of the doubly linked list. The square at the bottom right is the magnified view of a node in the doubly linked list.

Question 1: Basic Functionality

USE the User class developed by you in the previous Assignment

1. ***Create a class User to maintain User information***
2. ***Create a class UserList to maintain a doubly linked list of user records.***
3. ***Create an index using the Hashing technique on the attribute UserID.*** **For collision resolution, use the DoubleHAshing technique; for the hash function, use the Universal Hash Function discussed in class.**

Provide the following functions in the User List class:

1. Insert User -- A new user record is added to the User doubly linked list in sorted order efficiently using the AVL tree index. Also, update the AVL tree (index).
2. Search User -- Given a UserID, efficiently search using the AVL tree index and display the user record
3. Delete User -- Given a UserID, delete the user record from the AVL tree and the User List.
4. Print all Users

Ensure Insert, Delete, and Search operations are performed in O(logN) worst-case time.

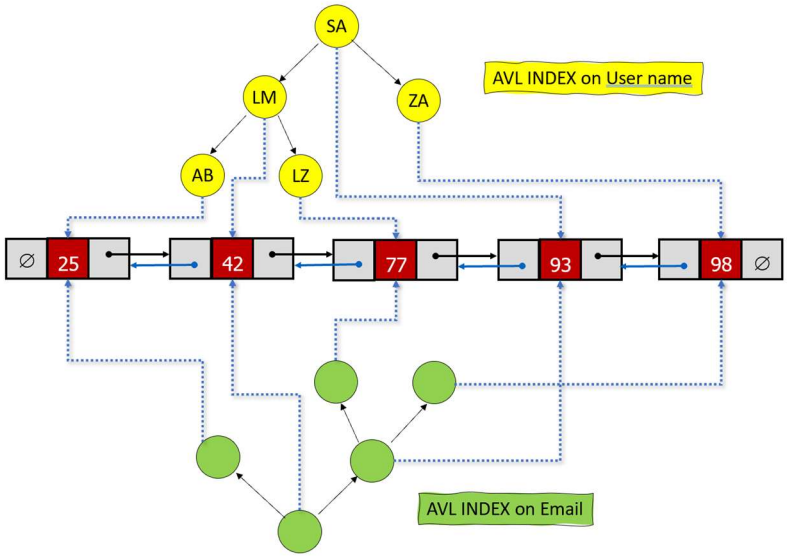
Question 2: More HASH Indices

Your program will also provide advanced functionality: creating indexes on different fields in user records for efficient searching. Each Index is created using **HASHING Technique.**

Provide the following functions

4. **Create an Index on the User name using the hashing technique. For collision resolution, use Chaining, and for the hash function, use a mix of the techniques discussed in class for the hash function.**
5. Delete an Index on the User name
6. Search the user **efficiently** using the user name and print his information.
7. **Create an Index on Email using the hashing technique. Use the Chaining technique for collision resolution, and use a mix of the techniques discussed in class for the hash function.**
8. Delete an Index on Email
9. Search the user **efficiently** using the user's email and print his information.
10. Display the Indices created on email and username.

Search functions for username and email will check if an index is available for the specified fields. If an index exists, the search will be performed using the AVL index in **O(log n)** time. If no index exists, the search will be conducted in a doubly linked list, resulting in **O(n)** time complexity.



The above figure shows two indexes on the user list, one on username and the other on user email. Hence, multiple indices can exist on the user list in addition to the AVL index on UserID. The data in the doubly linked list is sorted on the basis of UserID.

Question 3: Group HASH Index

Create an AVL-tree-based group Index. In Group Index, each AVL tree node can hold pointers to multiple nodes in the doubly linked list of Users. In other words, Each AVL node in the group Index contains a singly linked list of pointers to the User List

Provide the following functions

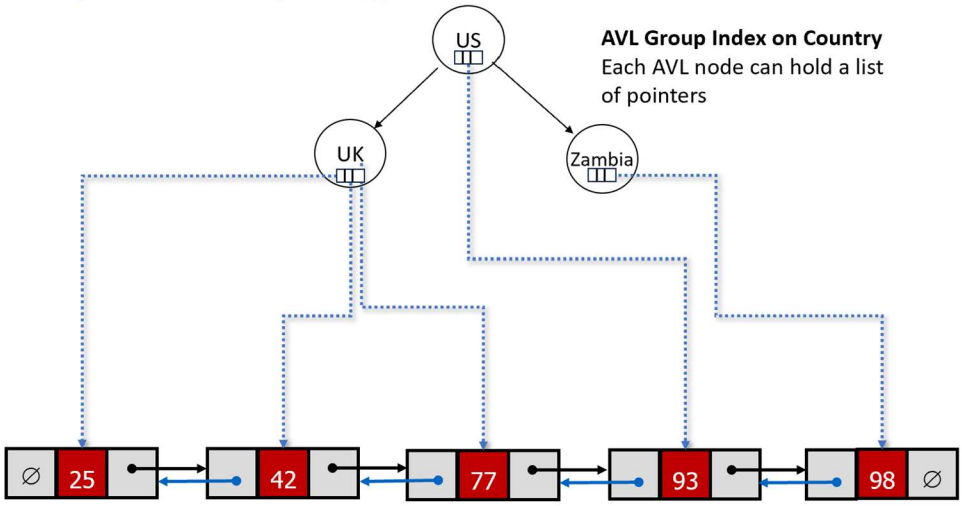
11. Create a group Index on the Country **using the hashing technique. For collision resolution, use Linear Probing , and for the hash function, use a mix of techniques discussed in class.**

1. Delete a group Index on the Country
2. List users that belong to a given Country
3. Display the group Index created for the country.

12. Create a group Index on the Type **using the hashing technique. For collision resolution, use quadratic probing, and for the hash function, use a mix of techniques discussed in class.**

4. Delete a group Index on the Type
5. List users that belong to a given Type
6. Display the Index created on the Type.

The index created on Country and Type differs from the AVL index on User ID. It is a group index, as many users can belong to one country or Type.



NOTE: Each insertion in the user list will also alter all the existing indices.

You can use the Standard template library (STD) vectors and doubly linked list for Hashing

Question 4: Registering Complaints

The following module should work with the Hashing Indices developed in previous question

Complaints are maintained using a priority queue. The system gives priority to users based on their type (platinum, gold, silver, new, regular). For users of the same type, complaints are registered on a first-come, first-served order. Hence, user Type and Complaint ID help us determine the next complaint to service.

Each complaint has a unique complaint ID, user ID (who filed the complaint), and text describing the complaint and issues faced by the user. *Hint: Create a class to hold information about the complaint.*

The priority queue will be implemented using heaps, which are implemented using an array, as discussed in class.

Provide the following functions for registering and maintaining complaints.

1. Register Complaint -- insert a complaint in the priority queue based on **user type.** For users of the same type, complaints are registered on a first-come, first-served order using the complaint ID.
2. Service Complaint -- process and remove the highest priority complaint.
3. Increase Priority – this function takes a pointer to the complaint in the priority queue and increases its priority to the top while maintaining the heap structure.
4. Display complaints registered by a given User ID. To ***efficiently*** search for the complaints registered by a user, create a group index using the AVL tree on the priority queue.
5. Print complaints registered by users of the given country **efficiently.** *Use the group index on the country created in the previous question to find users for the given country. Then, use the group index on UserID on the priority queue to print complaints registered by each user of that country.*