Bank Management System

Data Structures and Algorithms

Mohammad Abdur Rehman
2022299
Cybersecurity

Submitted to:

Dr. Ali Imran Sandhu

Table Of Contents

Introduction

A sophisticated tool called the Bank Management System was created to make handling customer accounts in a banking setting easier. This system makes use of sophisticated data structures, such as hash tables and Binary Search Trees (BSTs), to make it easier to store, retrieve, and manage customer information and transaction histories.

Objectives

The primary objectives of this system are:

- Efficiently store and manage customer details, including name, account number, address, and contact information.
- Provide rapid retrieval of customer information based on account numbers.
- Maintain a comprehensive transaction history for each customer account.
- Offer functionalities for adding accounts, deleting accounts, adding transaction details, and viewing customer information.

System Design

Data Structures:

- **Binary Search Tree (BST):**
- The BST is utilized to store customer information where the account number serves as the key for each node.
- This data structure facilitates efficient searching, insertion, and deletion operations based on account numbers.

```
Node* insert(Node* node, int key, customerinfo* customer)
  {
     if (!node) return new Node(key, customer);

     if (key < node->key)
        node->left = insert(node->left, key, customer);
     else if (key > node->key)
        node->right = insert(node->right, key, customer);

     return node;
  }

  customerinfo* searchBST(Node* node, int key)
  {
     if (!node || node->key == key) return (node ? node->customer : NULL);

     if (key < node->key)
        return searchBST(node->left, key);
     else
        return searchBST(node->right, key);
  }
```

- **Hash Table:**
- A hash table is employed to maintain pointers to customer information.
- The hash function maps the account number to an index within the hash table, enabling rapid access and management of customer data.

```
customerinfo* head;
   customerinfo* hashtable[10];
   Node* root;

   int hashFunction(int accno)
   {
      return accno % 10;
   }
```

Components:

- **customerinfo Structure:** Represents individual customer information and includes attributes such as name, account number, address, contact, and transaction history.

```
struct customerinfo
{
   string name;
   int accno;
   string address;
   int contact;
   customerinfo *next;
   customerinfo *prev;
   queue<string> transactionHistory;

   customerinfo(string name, int accno, string address, int contact) {
      this->name = name;
      this->accno = accno;
      this->address = address;
      this->contact = contact;
      next = NULL;
      prev = NULL;
   }
};
```

- **Node Structure:** Represents a node in the BST, containing a key (account number) and a pointer to customer information.

```
struct Node
{
   int key;
   customerinfo* customer;
   Node* left;
   Node* right;

   Node(int key, customerinfo* customer) : key(key), customer(customer), left(NULL),
right(NULL) {}
};
```

- **accountbook Class:** Encapsulates methods for managing customer accounts, including adding accounts, deleting accounts, adding transaction histories, and retrieving customer details.

Functionalities

Add Account:

- Allows users to input customer details, including name, account number, address, and contact.
- Inserts the customer information into both the BST and the hash table for efficient management.

```
void addAccount(string name, int accno, string address, int contact)
    {
      customerinfo* customer = new customerinfo(name, accno, address, 
contact);
      root = insert(root, accno, customer);

      int hashValue = hashFunction(accno);
      if (head == NULL) {
         head = customer;
      } else {
         customerinfo* temp = head;
         while (temp->next != NULL) {
            temp = temp->next;
         }
         temp->next = customer;
         customer->prev = temp;
      }
      customer->next = hashtable[hashValue];
      hashtable[hashValue] = customer;
    }
```

Delete Account:

- Enables users to delete a customer account based on the account number.
- Removes the corresponding customer information from the BST and hash table.

```
void deleteAccount(int accno)
    {
      int hashValue = hashFunction(accno);
      customerinfo* temp = hashtable[hashValue];

      while (temp) {
         if (temp->accno == accno) {
            if (temp->prev) {
               temp->prev->next = temp->next;
            } else {
                hashtable[hashValue] = temp->next;
            }

            if (temp->next) {
                temp->next->prev = temp->prev;
            }

            delete temp;
            cout << "Account deleted successfully!" << endl;
            return;
        }
        temp = temp->next;
    }
    cout << "Account not found!" << endl;
}
```

Add Transaction:

- Permits users to add transaction details for a specific account.
- Updates the transaction history queue associated with the respective customer account.

```
void addTransaction(int accno, const string& transaction)
  {
    int hashValue = hashFunction(accno);
    customerinfo* temp = hashtable[hashValue];

    while (temp) {
        if (temp->accno == accno) {
            temp->transactionHistory.push(transaction);
            return;
        }
        temp = temp->next;
    }
    cout << "Account not found!" << endl;
  }
```

View Customer Details:

- Retrieves and displays customer details based on the provided account number using the BST.

```
void getCustomer(int accno)
  {
    customerinfo* found = searchBST(root, accno);
    if (found) {
      cout << "Customer found with Account Number: " << found->accno <<
endl;
      cout << "Name: " << found->name << endl;
      cout << "Address: " << found->address << endl;
      cout << "Contact: " << found->contact << endl;
    } else {
      cout << "Customer with Account Number not found." << endl;
    }
  }
```

View Transaction History:

- Displays the transaction history associated with a specific customer account.

```
void printTransactionHistory(int accno)
  {
    int hashValue = hashFunction(accno);
    customerinfo* temp = hashtable[hashValue];

    while (temp) {
      if (temp->accno == accno) {
        queue<string> tempQueue = temp->transactionHistory;
        cout << "Transaction history for Account No. " << accno << ":" << endl;
        while (!tempQueue.empty()) {
          cout << tempQueue.front() << endl;
          tempQueue.pop();
        }
        return;
      }
      temp = temp->next;
    }
    cout << "Account not found!" << endl;
  }
```

Print All Customers:

- Prints a list of all customers with their respective details stored in the system.

```
void printList()
  {
    customerinfo* temp = head;
    while (temp != NULL) {
      cout << temp->name << " " << temp->accno << " " << temp->address << " "
<< temp->contact << endl;
      temp = temp->next;
    }
  }
```

Main Function:

```
int main()
{
  accountbook a1;
  int choice;
  int accno;
  string name, address, transaction;
  int contact;

  do {
    cout << "\n===== Account Management System =====" << endl;
    cout << "1. Add Account" << endl;
    cout << "2. Delete Account" << endl;
    cout << "3. Add Transaction" << endl;
    cout << "4. View Customer Details" << endl;
    cout << "5. View Transaction History" << endl;
    cout << "6. Print All Customers" << endl;
    cout << "7. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
      case 1:
        cout << "Enter name: ";
        cin.ignore(); // to clear the buffer
        getline(cin, name);
        cout << "Enter account number: ";
        cin >> accno;
        cout << "Enter address: ";
        cin.ignore();
        getline(cin, address);
        cout << "Enter contact: ";
        cin >> contact;
        a1.addAccount(name, accno, address, contact);
        break;

      case 2:
        cout << "Enter account number to delete: ";
        cin >> accno;
        a1.deleteAccount(accno);
        break;

      case 3:
        cout << "Enter account number for transaction: ";
        cin >> accno;
        cout << "Enter transaction details: ";
        cin.ignore();
        getline(cin, transaction);
        a1.addTransaction(accno, transaction);
        break;

      case 4:
        cout << "Enter account number to view details: ";
        cin >> accno;
        a1.getCustomer(accno);
        break;

      case 5:
        cout << "Enter account number to view transaction history: ";
        cin >> accno;
        a1.printTransactionHistory(accno);
        break;

      case 6:
        cout << "\n===== List of Customers =====" << endl;
        a1.printList();
        break;

      case 7:
        cout << "Exiting the program. Goodbye!" << endl;
        return 0;

      default:
```

```
                cout << "Invalid choice. Please enter a valid option." << endl;
                break;
        }

    } while (choice != 7);

    return 0;
}
```

Conclusion

The Bank Management System offers a robust and efficient solution for managing customer accounts and transaction histories within a banking environment. By leveraging advanced data structures and functionalities, the system ensures seamless operations, rapid data retrieval, and enhanced customer service. Future enhancements may include the integration of additional banking functionalities, security enhancements, and user-friendly interfaces to further optimize system performance and user experience.