DSA MiniProject Report

Group 14

Part A: Data Structures Used:

1) User Database: We have used a **hash table** to store the user data. The position of each user in the table is determined by their unique UID/
2) Bitcoin Block-Chain: We have implemented the block-chain using a **doubly linked list**. There are nodes at the start and the end to keep track of the length of the chain and also allow us to add blocks at the end in constant time.
3) Transaction History : A **static array** to store the transaction history of each user. The transactions are stored as and when they occur, in chronological order.

Part B: Algorithms (Some Functions we've implemented)

1) Hash value of the blocks: O(1)

A hash function that uses a long string of preset random characters and the information stored in the block (like the nonce, block number, transactions, etc) to calculate its hash value. This value is used to validate the block and restore corrupted information in case of an attack (nonce in our case).

2) Create Block : O(b) where 'b' is the number of blocks

This takes non-constant time because we call Validate Block Chain to validate the block-chain before adding another block.

3) **Attack : O(1)**

   Randomly generates a number between 1 and 50 and attacks the block with that index by changing its nonce.

4) **Validate Block Chain: O(b) where 'b' is the number of blocks**

   It starts traversal from the last block that was added at the end of the block-chain and compares the hash value of the previous block which has been stored (in the present block) with the recalculated hash value.
   If it turns out to be different, it concludes that the previous block was attacked (i.e nonce has been changed). Then, it proceeds to restore the original nonce by recalculating and comparing the hash value using all possible values of nonce.

5) **WriteBlocksToFile(): O(b)**

   Like PutUserHash(), this function writes the block chain to backup file block.txt just before main.c terminates so that the data can be used in subsequent runs of the program.

6) **GetBlocksFromFile(): O(b)**

   Like GetUserHash(), this function gets the information of the block chain from block.txt and uses it to create a block chain. The function is run in the start of main.c.

7) **Add User: O(1)**

   The function generated a unique ID for the new user (uniqueness is tested in constant time). The date and time of user creation and initial balance are stored in the user database userHash.

8) Transact: O(b) where ‘b’ is number of blocks

Takes two user IDs and the amount to be transferred. Checks the validity of the transaction, updates users’ transaction histories and adds a new block if applicable (this is where the complexity becomes O(b)).

9) GetUserHash(): O(n) where ‘n’ is number of users

Initializes a file stream to read users from the backup file user.txt one by one and restore them to userHash. This allows us to retrieve data from previous runs of the program. This is run at the start of main.c

10) PutUserHash(): O(n)

Initializes a file stream to put users from userHash to the file user.txt for access in the next run of the program. This is run just before main.c terminates.

Part C: Division of Work

Ananya Sane
2020102007

- GetBackupVsRefresh(): Lets user pick whether to restore data from user.txt and block.txt
- WriteBlocksToFile()
- GetBlocksFromFile()
- User.h: Creation of all structures required to store user data.
- initHash(): Initialize user database with all index IDs = -1
- makeID(): Generating unique ID for user
- setDate(): Get system date to store as user creation date
- setTime(): Get system date to store as user creation date
- makeUser(): Put all the components together, get user’s initial balance, and add new user to database

- GetUserHash()
- PutUserHash()
- README.md

M Krishna Praneet
2020113010

- Init_block_chain() : initialises the block chain with the head
- add_block() : to add a block (create block)
- validate_block_chain() : validates the block-chain
- makeID() : generating unique ID for user
- Block.h : declare functions and structs
- update_trav() : updates the transactions in the latest block and calls add_block when needed
- README.md

Trisha Kaore
2020111021

- main.c : The program which includes all the other .c files, and acts as the user interface.
- transaction() : lets the user make transactions.
- printHistory() : prints a user's transaction history.
- updateHistory() : updates the sender's and receiver's transaction history.
- create_users.c : program which generates 'UserNum' number of users with balances which will be added in the main program.
- printBlockchain() : prints the chain of blocks.
- printUsers() : prints the already existing users.
- Helped in user.h.

G Vinaychandra Reddy

2020101010

- Functions:
- attack()
- Hash function()
- add_block()
- Helped in main.c

B Vishwateja Reddy

2020102058

- Init_block_chain() : initialise the block chain with the head
- add_block() : to add a block
- Block.h : declares functions and structs
- Helped in main.c