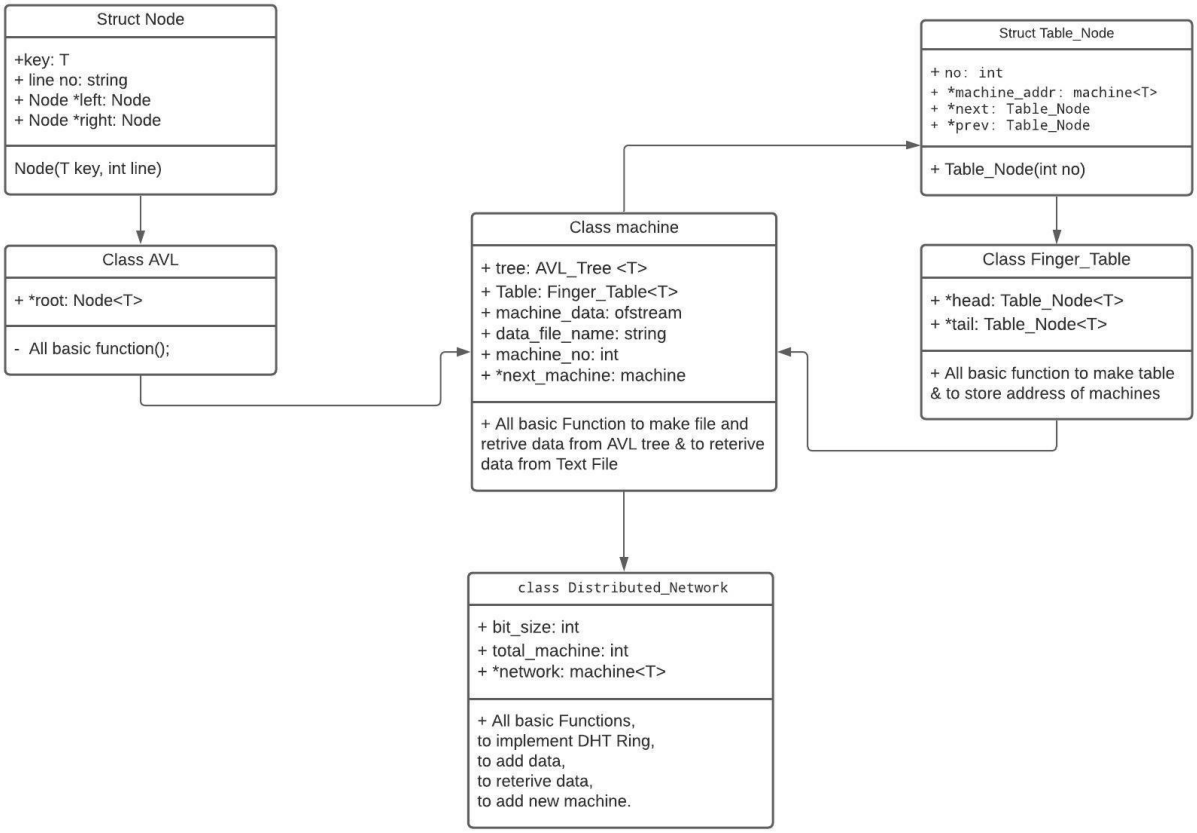
DS Project Report

Group Members

- Hassan Ashfaq {19I-1708}
- Sheeza Azher {19I- 1878}
- Saif Sadiq {19I-1877}

Our program is to consider a scenario where the data is not located on one singular machine but stored in multiple machines distributed across the Internet. Since retrieving data from such a cloud of information is difficult, therefore we use a data structure called a Distributed Hash Table. It provides us the ability to search for information efficiently using a key and value method (retrieving value of machine using a given key).

Our Project UML Diagram



- Total Classes:
    - Class AVL_Tree
    - Class Finger_Table
    - Class Machine
    - Class Distributed_Network

Our main class is Distributed_Network, all classes are connected/linked as shown in the above UML Diagram. All Class are briefly explain below with the members and function respective function.

Library Used for Hashing: - #include <functional>

Class Distributed_Network

Class Distributed_Network:

- Private:
  - int bit_size: the number of bits given by the user
  - int total_machine: number of machines in the network
  - machine* network: Head node for the circular linked list for machine
  - void sort (): Function to sort the circular linked list
- Public:
  - Default Constructor
  - Setters
  - Void make_network_using_IP (T no): This function links machines using their IP address {IP address is hash into int } and handles adding machine into the network and makes file of the machine after making the distributed network.
  - Void make_network_using_machine_no (int no): This function links machines using their machine number and handles adding machines in the network and makes file of the machine after making the distributed network.
  - Void insert_data_to_cloud (T val, T data, int k): This function inserts a key, value pair into the distributed network in any machine.
  - Void display_all_machines (): This function displays all machines in the distributed network along with their address.
  - Void manage_table (): This function creates a finger table for each machine.
  - Void display_table_of_machines (int a): This function displays the finger table of the machine using machine number.
  - Void display_AVL_of_machines (int a): This function displays AVL tree using machine ID/no.
  - T reterive_data_from_cloud (T key): This function retrieves data from distributed network using the key.
  - T reterive_data (T key, int no): This function retrieves data from the machine using the machine ID given by the user.
  - Void delete_machine_from_cloud (int id): This function deletes machine using machine ID and stores it AVL tree in the next machine.
  - Void delete_data (T key, int k): This function deletes the key-value pair from the machine from any machine.

Class Machine

Class machine:

- Private :
  - AVL_Tree <T> tree : The AVL tree for the machine
  - Finger Table<T> Table : The finger table for the machine
  - Ofstream machine_data: object of ofstream library
  - T data_file_name: file for storing the machine's information
  - Int counter: integer to make extra file if data is great than 100 in file {data_file_name_counter.txt} i.e. machine2_counter.txt where counter = 0 to n

- Public:
  - int limit: limit of the data stored in a file
  - Int machine no: integer for storing machine ID
  - machine*next machine: Pointer to make machine link list
  - Constructor
  - Setters
  - Void make file (): this function makes a unique file for each machine.
  - Void make_extra_file (): This function makes an extra file if there were more than 100 lines of data in the first file of the machine.
  - Void insert(T val, T data) : This function inserts a key, value pair into the Machine
  - T retrieve (T val): This function retrieves data from the machine.
  - Void make_Finger_Table (int n, machine<T>*list): This function makes a finger table for the machine.
  - Void display_table(int a): This function displays the finger table for the machine
  - Void swap_data (machine<T>*temp, machine<T>*temp1): This function swaps the data between two machines.
  - machine<T>* reterive_using_lookup(int n): This function retrieves the machine pointer from the finger table.
  - Void print_AVL (): This function prints the AVL tree.
  - Void shift_AVL_Tree_Data (machine<T>*a): This function shifts the AVL tree from 'a' to another tree.
  - Void delete_data_from_AVL_Tree (T key): This function deletes AVL Tree Node using a key.

Class Finger_Table

Class Table_Node:

- Data members:
  - Int no : Machine number
  - Machine<T>*machine_addr : Address of machine
  - Table_Node*next : Stores address of next node
  - Table_Node*prev : Stores address of previous node
  - Constructor

Class Finger_Table:

- Private:
  - Table_Node<T>*head : head of the finger table list
  - Table_Node<T>*tail: tail of the finger table list

- Public member:
  - Constructor
  - Void make_table (int n, machine<T>*machines, int machine no): This function makes the finger table for each machine.
  - Void display(int a): This function displays the finger table
  - Machine<T>*retrieve(int no): This function retrieves machine pointer from the machine
  - Destructor

Class AVL_Tree

Struct Node:

- Data members:
  - T key: value in the node
  - Int line_no: Line number where the node is stored in the file
  - T file_name: File name of the file of the node
  - Int height: height of the node
  - Node*left: Address of the left node
  - Node*right: Address of the right node
  - Constructor

Class AVL_Tree:

- Private:
  - Node<T>*root : root node of the tree
  - Node<T>*insert_data (Node<T>*root, T val, int line_data, T file): This is recursive function to insert data into the nodes and balance the tree.
  - Int height (Node<T>*root): This function returns the height of the tree at any node given.
  - Node<T>* Single_Right_Rotate (Node<T>*&root): This is the function for a single right rotation.
  - Node<T>*Single_Left_Rotate (Node<T>*&root): This function is for single left rotation.
  - Node<T>* Double_Left_Rotate (Node<T>*& root): This function is for double left rotation.
  - Node<T>* Double_Right_Rotate (Node<T>*& root): This function is for double right rotation.
  - void inorder (Node<T>* root): This function is to print inorder traversal of the AVL Tree.
  - Node<T>* MIN (Node<T>* root): This function return the minimum value node in the left subtree.
  - Node<T>* MAX (Node<T>* root): This function returns the maximum value node in the right subtree.
  - Node<T>* remove (Node<T>* root, T val): This is a recursive function to remove a node with the value 'val' and balances the tree.
  - void Search(Node<T>* root, T val, int& data): This function searches the tree using a key and return the line number on which it is saved.
  - void Search_File (Node<T>* root, T val, T& data): This function searches the tree using a key (val) and return the file name.
  - void shift_data (Node<T>* root): This function shift the data of the AVL tree of the machine into this AVL tree.

- Public:
  - Constructor
  - Void insert (T val, int line_data, T file): This function inserts values into the tree.
  - Void remove_node (T val): This function removes node with the value 'val'.
  - Void display (): This function displays the inorder traversal of the tree.
  - Int Find (T val): This function return the line the 'val' is stored on.

- T Find_File (T val): This function returns the file ‘val’ is stored on.
- Void shift_data_from_different_AVL_Tree (AVL_Tree<T> a): This function shifts data from another AVL_Tree to this AVL_Tree.
- Destructor