**DISTRIBUTED HASH SYSTEM**

REPORT BY

(Umair Afzal 19I-0517)

We’ll explain the basic purpose of all functions and classes below and their relationship with each other.

1. **DHT.h:**

We have included two files in it.

* hash.h
* CircularLinkedList.h
* **ComputeShahash function:**

This function is being used to get the hash value of the string that we are passing. We have used the functions of SHA1 class in this hash function.

* **hexadecimalToDecimal function:**

This function, as the name specifies, is used to change the hexadecimal values in decimal values.

* **Class DHT:**
* **Createidentifierspace:**

This function is used to specify the size of identifier space in bits.

* **Specifynumberofmachines:**

This function is used to specify the number of machines and also gives the choice to user to enter id of machines manually or automatically. And at last it adds the id as specified by user i.e. manually or automatically.

It then inserts that machine’s id in the circular linked list.

* **Addmachine:**

This function adds a new machine on the fly without disrupting the functionality of Ring DHT. It also inserts the machine’s id in our circular linked list.

* **Search:**

This function basically finds the machine which is responsible for data through routing tables and then return the pointer to that machine.

* **Adddata:**

This function inserts the data in the form of a key from any machine. Then it stores the data at the specified machine through hashing and also tells the path where the data is stored by calling the search function in it.

* **Redestributedata:**

This function basically redistributes the data after every new machine is added in the system.

* **Removemachine:**

This function ask user to enter an id and then remove that machine with the same id if a machine with the given id exists. After removing the machine, if there was some data that was being stored on the specified machine (which we removed), then the data is first stored in a queue and then the machine is removed. And after that the data is again inserted in the system among the remaining machines left. And in this way a machine gets removed without disturbing the whole DHT ring.

* **Adddataagainafterremoval:**

This function adds the data again in the system, on the specified machine by using hashing, after a machine has been removed from the system.

* **Removedata:**

As the name specifies, this function is used to remove the data from the system. After removal it also tell the path from where data is removed.

* **Printavltree:**

This function is self-explanatory. It prints the avl tree of a specified machine on the screen.

* **Printroutingtable:**

This function prints the routing table of the machine whose id is given by the user.

* **Succ:**

It finds the successor of a value that which machine is responsible for it.

* **Constructroutingtables:**

This function is making routing tables by using doubly linked list and also by using the formula that is given to us in the pdf.

* **Menu:**

This function is displaying the menu. First of all, this function is called. It gives user different options to select from. After selecting a specific option, it calls the required functions that are to be used to fulfil user requirements.

1. **DoublyLinkedList.h:**

We have included CircularLinkedList.h in it.

This is basically a double linked list in which we have included the following functions:

* **Insert:**

This function inserts the values in the list and maintains the next and previous pointers as it is a doubly linked list.

* **Remove:**

This function removes a particular node from the list and rearrange the next and previous pointers.

* **Display:**

This function displays all the contents of the list.

1. **CircularLinkedList.h:**

We have included the following files in it:

* AvlTree.h
* DoublyLinkedList.h

This file contains a circular linked list named as MachineList and its node struct named as machineNode which contains the following functions:

* **Search**

It tells if a specific id machine exists or not.

* **Insert**

It inserts the data in the list if we have identifier space else it tells us that the space is full and we can not enter any other data.

* **Remove**

It removes the specific id node from the list.

* **Display**

It displays the contents of the linked list. It displays machines’ id, address, tree data and routing table.

1. **AvlTree.h:**

We have included queue.h file in it. This file basically has the code of an AVL tree which contains the following functions:

* **Insert:**

Adds data in new node. Then checks if the tree is balanced or not. If the tree is balanced then our insertion is complete but if tree is not balance after insertion then this function makes it a balance tree (AVL Tree) by applying the correct rotation.

* **Remove:**

Deletes the specific node from the tree and checks if the tree is balanced after the removal. If the tree is balanced then our removal is completed but if it is not an AVL Tree then this function makes it into an AVL Tree (balanced tree) by applying the correct rotation.

* **Display:**

It displays the content (nodes) of the whole tree (inorder traversal).

* **Height:**

Returns the height of the tree.

* **SingleRightRotate:**

Rotates the tree to the right for once.

* **SingleLeftRotate:**

Rotates the tree to left for once.

* **DoubleRightRotate:**

First rotates to left and then rotates to right.

* **DoubleLeftRotate:**

First rotates to right and then rotates to left.

* **Findmin:**

Finds the leaf node at the left of the given tree which is a minimum value node.

* **Getroot:**

Gives the root of the tree.

* **Inorder:**

Traverses the AVL Tree in inorder traversal format.

* **Getdata:**

Enqueues the data in the queue.

1. **Queue.h:**

This file contains the code of a queue. The queue contains the following functions:

* **Front:**

Returns the front pointer of the queue.

* **Back:**

Returns the rear pointer of the queue.

* **Enqueue:**

Enters the data in the queue.

* **Dequeue:**

Removes the data that is at front from the queue.

* **Is\_empty:**

Tells if the queue is empty or not.

* **Display:**

Displays the contents of the queue.

* **Makeempty:**

This function empties our queue.

1. **Hash.h:**

This is a simple hash class that will give us the hash value of our string.

1. **Main.cpp:**

We have included DHT.h file in it.

* **main:**

We have not done much in main. We have made an object of DHT class in main and then we have called the menu function of DHT class.

**Please see below the class diagram of our whole project:**

**CLASS DIAGRAM**

