

GROUP MEMBERS :

TALHA RIZWAN I190652

MUHAMMAD IBTISAM I190422

ANSER QURESHI I190680

**DISTRIBUTED HASH TABLE PROGRAM**

**DISCRIPTION**

**MINOR Classes:**

* AVLTreenode
  + to store record of an inserted key in an AVLTree / machine.
  + The class also maintains the file path and line no in file it which the data is stored
  + Work being as a data member of AVLTree class
* NODE
  + maintaining nodes for routing table
  + works as data member of doblylinkedlist class
* nod
  + Possesses all data of a single machine.
  + Works as a data member of linkedlist class

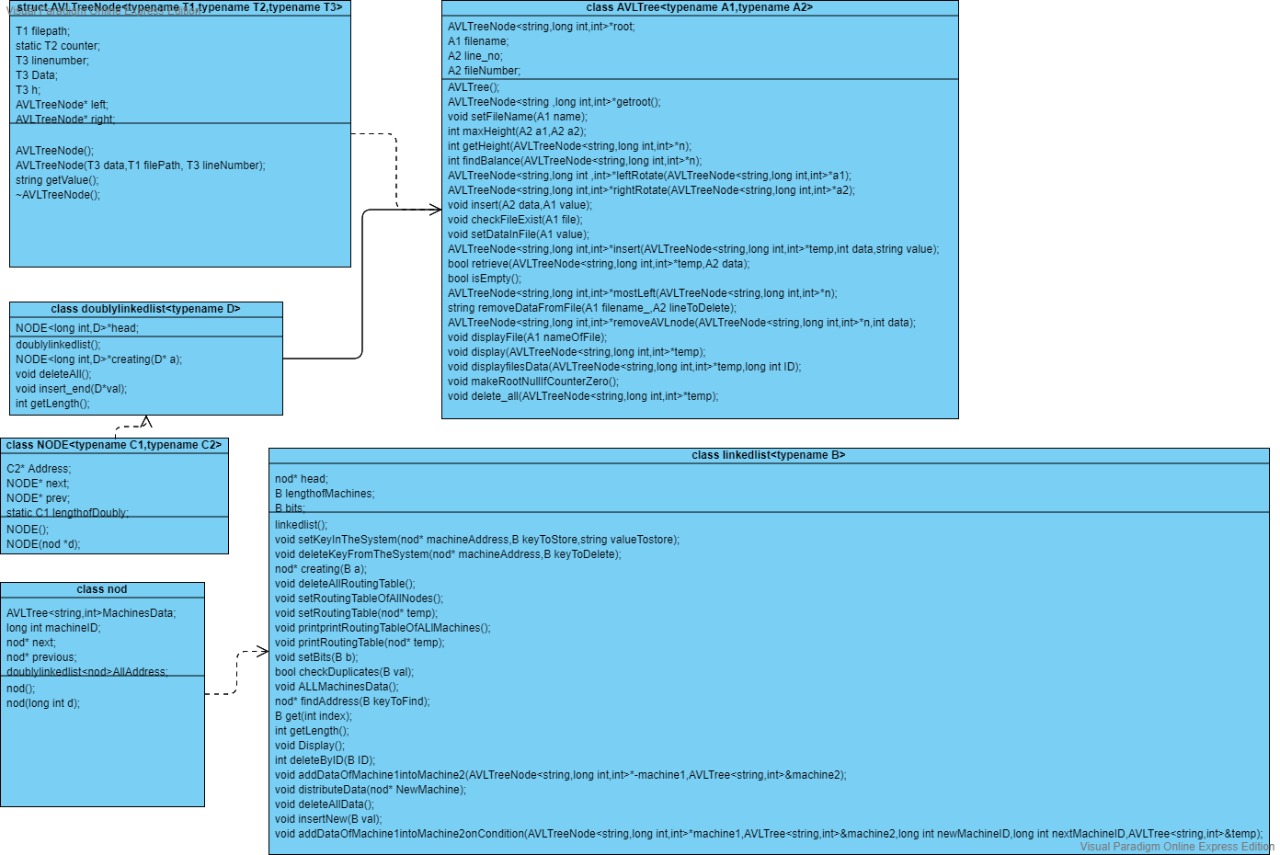
**MAJOR Classes:**

* AVL Tree class
  + including AVL Tree Node class via composition.
  + maintain tree by starting it from head.
  + organize all data of a single machine.
* doublylinkedlist
  + maintain data of all machines/nodes class.
  + Implemented on concept of doubly Circular linkedlist.
* linkedlist
  + maintaining routing table for different addresses of machines/nodes.
  + A doubly linked list and non-circular in nature.

**Note: all classes are template based.**

**Note: each class is in different file.**

**Class Diagram**



**Functions Of All Major Classes:**

**AVLTree class:**

* The templates used by program are A1=string and A2=integer

**+ setFileName(A1 name):void //it sets the file name for a single node to store data**

**+ maxHeight(A2 a1, A2 a2):int //tells which side has greater height**

**+ getHeight(AVLTreeNode\* n):int //returns height of an avl tree node**

**+ findBalance(AVLTreeNode\* n):int //returns balance of required node**

**+ rightRotate(AVLTreeNode\* a2): AVLTreeNode //rotate right in tree when needed**

**+ leftRotate(AVLTreeNode\* a2): AVLTreeNode //rotate left in tree when needed**

**+ insert(A2 data, A1 value):void //to insert a new node**

**+ checkFileExist(A1 file):void //to check for already existance of file**

**+ setDataInFile(A1 value):void //puts data in fie**

**+ retrieve(AVLTreeNode\* temp, A2 data): bool //search for a specific node in tree**

**+ isEmpty():bool //tells if the tree or root is empty**

**+ AVLTreeNode\* mostLeft(AVLTreeNode\* n) //returns the most smallest value in tree**

**+ removeDataFromFile(A1 filename\_, A2 lineToDelete):string //removes the data on a particular line no in the file of a tree record**

**+ removeAVLnode(AVLTreeNode n, int data): AVLTreeNode\* //removes a required node from tree**

**+ display(AVLTreeNode\* temp):void //display the tree**

**+ displayfilesData(AVLTreeNode\* temp, long int ID):void //display the data in tree**

**+ makeRootNullIfCounterZero():void //put root to null when called by delete\_all ftn**

**+ delete\_all(AVLTreeNode\* temp):void //deletes all nodes from tree and erase all data of a single machine**

**linkedlist class:**

* The template used by program is B=long integer.

**+ setKeyInTheSystem(nod\* machineAddress, B keyToStore, string valueTostore):void //specifying the machine from which we want to insert data**

**+ deleteKeyFromTheSystem(nod\* machineAddress, B keyToDelete):void //specifying the machine from which we want to delete data**

**+ deleteAllRoutingTable():void //function to delete all routing tables**

**+ setRoutingTableOfAllNodes():void //setting the routing tables of all machines**

**+ setRoutingTable(nod\* temp) :void //setting the routing table of a single node**

**+ printRoutingTable(nod\* temp):void //printing the routing table of a single machine**

**+ successor(B key):nod\* // return machine id address**

**+ checkDuplicates(B val):bool //return false if duplicate found**

**+ ALLMachinesData():void //prints all the data of all machines available**

**+ findAddress(B keyToFind):nod\* //it finds the address by specifying the key**

**+ Display():void //display the machine ids of all the machines**

**+ deleteByID(B ID):int //deletee by id;// return 0 when enter invalid id, 1 when no data, 2 when id found and deleted**

**+ addDataOfMachine1intoMachine2(AVLTreeNode\* machine1, AVLTree& machine2):void**

**//it recevies head of machine1 to be deleted and avl object of machine 2 in which data is to add**

**+ addDataOfMachine1intoMachine2onCondition(AVLTreeNode\* machine1, AVLTree& machine2, long int newMachineID, long int nextMachineID, AVLTree& temp):void**

**//first parameter : root of AVL tree of next machine of new machine//second parameter : AVL object to store data of new machine**

**//third parameter : new machine id//Fourth parameter : Machine id of the next machine from where data is getting**

**//fifth parameter : reference of avl object to store the rest of the data**

**//Description : split the data into new machine from next machine of the new machine**

**+ distributeData(nod\* NewMachine):void // parameter : pointer of new machine where data is to be stored from its next machine**

**//Description : split the data into new machine from next machine of the new machine**

**+ deleteAllData():void //it deletes all the data**

**+ insertnew(B val):void //it inserts in the list**

**doublylinkedlist class:**

* The template used by program is D=nod class

**+ deleteAll():void //to delete all nodes in the list**

**+ insert\_end(D\* val):void //inserting a new node in the list at the end**

**+ getLength():int //function to calculate the length of the list**

**Goals Achieved:**

**1.** Option to specify the number of machines.

**2.** Option to specify the size of identifier space in bits, i.e., 160 bits, 4 bits etc.

**3.** Option to manually assign Id to each machine. If manual Ids are not assigned by the user, the system must automatically assign Ids.

**4.** Option to insert data in the form of key, value pairs from any machine. The insertion must show the complete path taken by the request to arrive at the correct machine. Additionally, the option to print AVL tree.

**5.** Option to remove data by specifying the key from any machine. The output must show the complete path taken by the request to arrive at the correct machine and corresponding value that is removed from the AVL tree. The updated AVL tree must be printed.

**6.** Option to print the routing table of any machine.

**7.** Option to print the AVL tree maintain on any machine along with the location of files (and line numbers) on which the corresponding values are stored.

**8.** Option to add new machines on the fly without disrupting the functionality of Ring DHT.

**9.** Option to remove any machine on the fly without disrupting the functionality of Ring DHT



**THE END**