Table Of Content

Class Assignment3

```
java.lang.Object
    |
    +--Assignment3
```

< Constructors > < Methods >

public class **Assignment3**
extends java.lang.Object

driver class

Author:

Soham Gaikwad, 2018cs10394 all methods compile, no work is remaining

interesting Findings of assignment: if deletion is done by finding succeccor in RLLLL.... then when the case of same fname occurs, it must be inserted in the right else it can be the case that it is not searched.

Constructors

Assignment3

```
public  Assignment3()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

reads the file and give output acc to input query

Parameters:

args -

Class BST

```
java.lang.Object
    |
    +--BST
```

class **BST**
extends java.lang.Object

Fields

address

```
private static java.lang.String address
```
address of node

counter

```
private static int counter
```
count for operation i.e. no. of node touched

deleteError

```
private static boolean deleteError
```

insertError

```
private static boolean insertError
```

root

```
private BSTNode root
```
root of bst

Constructors

BST

```
BST()
```

Methods

delete

```
private BSTNode delete(BSTNode root,
                       java.lang.Comparable key)
```

---

deleteBST

```
boolean deleteBST(java.lang.Comparable key)
```

---

getAddress

```
java.lang.String getAddress()
```

Returns:

address of node

---

getCounter

```
int getCounter()
```

Returns:

count of operation // no, of node touched

---

inorderSuccessor2

```
private BSTNode[] inorderSuccessor2(BSTNode root)
```

find the inorder successor of root stores the successor and previous pointer in array res

Parameters:

root -

Returns:

res

---

insert

```
private BSTNode insert(BSTNode root,
                       java.lang.Object data,
                       java.lang.Comparable key)
```

insert data, key in subtree starting with root

which takes a root reference and compares the key of the root with the key of the given node to be inserted. if it the key of the node to be inserted is greater then the root key it calls recursive inserthelp on the right of the present root , this time sending root.right as root reference. if key is smaller then same operation is done on root.left

base case is when a null position is found, the new node is inserted there. since return type of this function is node the entire tree is modified after insertion.

Parameters:

root -
data -
key -

Returns:

root after inserting node

---

insertBST

```
boolean insertBST(java.lang.Object data,
                  java.lang.Comparable key)
```

insert new node with data and key in bst, calls insertBST

Parameters:

data -
key -

Returns:

true if no error on insert

---

search

```
private BSTNode search(BSTNode root,
                       java.lang.Comparable key)
```

compares the key of the root with the given key and accordingly decides whether to look in the left of the right sub tree. when key is matched the node is returnd. it also increments the update counter each time it touches a node.

Parameters:

root -
key -

Returns:

node if key is found in subtree at root , null otherwise

searchBST

BSTNode **searchBST**(java.lang.Comparable key)

caller function for searching key , calls searchBST

Parameters:

key -

Returns:

node with key

Class BSTNode

```
java.lang.Object
    |
    +--BSTNode
```

< Fields > < Constructors > < Methods >

class **BSTNode**
extends java.lang.Object

Fields

data

private java.lang.Object **data**
data or student in our case

key

private java.lang.Object **key**
hashed key

left

BSTNode **left**

right

BSTNode **right**

Constructors

BSTNode

```
BSTNode(java.lang.Object data,
        java.lang.Object key)
```

Methods

getData

```
java.lang.Object getData()
```

---

getKey

```
java.lang.Object getKey()
```

---

setData

```
void setData(java.lang.Object data)
```

---

setKey

```
void setKey(java.lang.Object key)
```

Interface Hash

< Methods >

---

public interface **Hash**

for hash functions h1 and h2

Methods

djb2

```
public static long djb2(java.lang.String str,
                        int hashtableSize)
```

h1

Parameters:

str -
hashtableSize -

Returns:

---

sdbm

```
public static long sdbm(java.lang.String str,
                        int hashtableSize)
```

h2

Parameters:

str -
hashtableSize -

Returns:

Class InvalidValueException

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--InvalidValueException
```

All Implemented Interfaces:
java.io.Serializable

< Constructors >

---

class **InvalidValueException**
extends java.lang.Exception

exception when -1 is returned by any of the fn

Constructors

InvalidValueException

```
InvalidValueException()
```

Class KeyPair

```
java.lang.Object
    |
    +--KeyPair
```

All Implemented Interfaces:
java.lang.Comparable

< Fields > < Constructors > < Methods >

---

public class **KeyPair**
extends java.lang.Object
implements java.lang.Comparable

Fields

first

```
private java.lang.Object first
```

---

second

```
private java.lang.Object second
```

Constructors

KeyPair

```
KeyPair(java.lang.Object first,
        java.lang.Object second)
```

Methods

compareTo

`public int **compareTo**(KeyPair xyKeyPair)`

compares based on first (firstname)

---

getFirst

`private java.lang.Object **getFirst**()`

---

toString

`public java.lang.String **toString**()`

returns fullname

Overrides:

toString in class java.lang.Object

Interface MyHashTable_

< Methods >

---

public interface **MyHashTable_**

Methods

address

`public java.lang.String **address**(java.lang.Object key)`

---

contains

`public boolean **contains**(java.lang.Object key)`

---

delete

```
public int delete(java.lang.Object key)
```

---

get

```
public java.lang.Object get(java.lang.Object key)
```

---

insert

```
public int insert(java.lang.Object key,
                  java.lang.Object obj)
```

---

update

```
public int update(java.lang.Object key,
                  java.lang.Object obj)
```

Class MyHashTable_BST

```
java.lang.Object
    |
    +--MyHashTable_BST
```

All Implemented Interfaces:
Hash, MyHashTable_

< Fields > < Constructors > < Methods >

---

public class **MyHashTable_BST**
extends java.lang.Object
implements Hash, MyHashTable_

class implementing separating chaining using bst param K key param T node / student

Fields

hashSize

```
private int hashSize
```

hashTable

`private BST[] **hashTable**`
size of the hashtable

Constructors

MyHashTable_BST

`MyHashTable_BST``(int hashSize)`

Methods

address

`public java.lang.String **address**(java.lang.Comparable key)`

searches the node with the given key present in the bst of hashtable[index] using the searchBST of bst class and returns it address else error .

TIME complexity of update in separate chaining:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the talbe size) then average case complexity is O(1+y) depending on n/N.

contains

`public boolean **contains**(java.lang.Comparable key)`

searches the node with the given key present in the bst of hashtable[index] using the searchBST of bst class and returns true iff it is not null .

TIME complexity of update in separate chaining:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the talbe size) then average case complexity is O(1+y) depending on n/N.

delete

```
public int delete(java.lang.Comparable key)
```

creates a index based on key and delete the that key from the bst present in hashtable[index] using deleteBST.

:TIME complexity of delete in separate chaining:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the talbe size) then average case complexity is O(1+y) depending on y.

---

get

```
public java.lang.Object get(java.lang.Comparable key)
```

searches the node with the given key present in the bst of hashtable[index] using the searchBST of bst class and returns it if it is not null else throws error.

TIME complexity of update in separate chaining:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the talbe size) then average case complexity is O(1+y) depending on n/N.

---

insert

```
public int insert(java.lang.Comparable key,
                  java.lang.Object obj)
```

jimplements insertBST of bst class creates a node based on key and object and generates a index using djb2 and inserts this node to hashtable[index] and returns the number of nodes touched in delete. Time complexity:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the table size) then average case complexity is O(1+y) depending upon n/N.

---

update

```
public int update(java.lang.Comparable key,
                  java.lang.Object obj)
```

searches the node with the given key present in the bst of hashtable[index] using the searchBST of bst class and then updates it .

TIME complexity of update in separate chaining:

a.worst case:O(n) where n is the number of keys and this happens when all keys are hashed to the same cell and the tree created is such that its height is n-1.

b.best case:O(1) when no collision occurs.

c.average case.:hash is generated in O(1) time and if y is the load factor(=n/N where n is the number of keys present in the hash table and N is the talbe size) then average case complexity is O(1+y) depending on n/N.

Class MyHashTable_DH

```
java.lang.Object
    |
    +--MyHashTable_DH
```

All Implemented Interfaces:
Hash, MyHashTable_

< Fields > < Constructors > < Methods >

---

public class **MyHashTable_DH**
extends java.lang.Object
implements Hash, MyHashTable_

Fields

hashSize

```
private int hashSize
```

stores the hash table size

---

hashTable

```
private java.lang.Object[] hashTable
```

this is the hashtable on which all operations are performed.

Constructors

MyHashTable_DH

```
MyHashTable_DH(int hashSize)
```

Methods

address

```
public java.lang.String address(java.lang.Object key)
```

//works in the same way as contains.

"E" is printed when object is not present.

Time complexity:

a.best case:O(1)//when the match is found at the very first index that is calculated.

b.worst case:O(n)//when we need to rehash almost every time and thus probe through all elements

c.avg case:O(1).

Parameters:

key -

Returns:

address/index of node of given key

---

contains

```
public boolean contains(java.lang.Object key)
```

//works in the same way as delete,update i.e continues to generate new index until a match is found , then updates

Time Complexity:

a.best case:O(1)//when object to be searched is present at the very first index that is generated.

b.worst case:O(n)//when we need to generate new index almost everytime .

c.averge case:O(1)

Parameters:

key -

Returns:

true if node with key is found , false otherwise

---

delete

`public int delete(java.lang.Object key)`

/deletes an entry from the hashtable when an objext with the given key is found. //calculates an index based on key and matches the with key of the element present at the index when a match is found it changes the status of that pair object in hashtable to false and thus marks it as deleted. //prints E when the object to be deleted is not present or is already deleted.

T IME COMPLEXITY:

best case:O(1)//when object to be deleted is found at the very first index that is calculated.

worst case:O(n)//when we need to probe through all the elements and then the object to be deleted is found

avg case:O(1)

Parameters:

key -

Returns:

no of operation req to delete if possibel , -1 otherwise

---

get

`public java.lang.Object get(java.lang.Object key)`

returns the object with the given key to the driver class and the driver class

//works similar to contains.

//prints E when object is not present.

Time complexity:

a.best case:O(1)//when the match is found at the very first index that is calculated.

b.worst case:O(n)//when we need to rehash almost every time and thus probe through all elements

c.avg case:O(1).

Parameters:

key -

Returns:

node with given key works similar to contains.

---

insert

```
public int insert(java.lang.Object key,
                  java.lang.Object obj)
```

inserts new node with key , and value obj t first generates an index using hash function h1(djb2) based on the concatenation of first+last name of the student, if it no entry is present at the calculated index or the entry present at that index was deletedthen it inserts it at the given index otherwise it generates a new index(index=h1+i*h2, where i=0,1,2..) using hash function h2(sdbm).this process ofgenerating a new index is continueduntil a null position is found in the hash table. or no. of iterations (i) becomes equal to hashSize (as once i = hashSize we get newHashedKey same as the initial hash key

TIME COMPLEXITY(n is no. of nodes or no. of entries):

best case:O(1)// when no collision occurs.

worst case:O(n)// when hashtable is almost full and collision occurs almost every time and we need to probe over all n elements

average case:O(1).

Parameters:

key, - obj

Returns:

no. of operations needed to insert if inserted, -1 otherwise

---

update

```
public int update(java.lang.Object key,
                  java.lang.Object obj)
```

find the object with the given key in hashtable and repaces it with the given obj. variable present continues to probe until the object with the given key is found and then replaces with new pair

Time compelexity:

a.best case:O(1)//when th entry to be updates is present at the very first index that is calculated.

b.worst case:O(n)//when we need to probe through almost all elements to find the entry to be updated

c.avg case:O(1);

Parameters:

key, - obj

Returns:

no. of operation needed to update if updated or -1 otherwise

Class NotFoundException

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--NotFoundException
```

All Implemented Interfaces:
java.io.Serializable

< Constructors >

---

class **NotFoundException**
extends java.lang.Exception

Constructors

NotFoundException

`NotFoundException``()`

Class Student

```
java.lang.Object
    |
    +--Student
```

All Implemented Interfaces:
Student_

< Fields > < Constructors > < Methods >

---

public class **Student**
extends java.lang.Object
implements Student_

Fields

cgpa

`java.lang.String` **`cgpa`**

department

java.lang.String **department**

fname

java.lang.String **fname**

hostel

java.lang.String **hostel**

lname

java.lang.String **lname**

Constructors

Student

```
public  Student(java.lang.String fname,
                java.lang.String lname,
                java.lang.String hostel,
                java.lang.String department,
                java.lang.String cgpa)
```

Methods

cgpa

public java.lang.String **cgpa**()

department

public java.lang.String **department**()

equals

`public boolean **equals**(java.lang.Object o)`

compares fname+lname

Parameters:

o - o will be key

Returns:

true if fullname is equal , false otherwise

Overrides:

equals in class java.lang.Object

---

fname

`public java.lang.String **fname**()`

---

hostel

`public java.lang.String **hostel**()`

---

lname

`public java.lang.String **lname**()`

---

toString

`public java.lang.String **toString**()`

Overrides:

toString in class java.lang.Object

Interface Student_

< Methods >

---

public interface **Student_**

Methods

cgpa

```
public java.lang.String cgpa()
```

---

department

```
public java.lang.String department()
```

---

fname

```
public java.lang.String fname()
```

---

hostel

```
public java.lang.String hostel()
```

---

lname

```
public java.lang.String lname()
```

INDEX

S

T

U