Gebze Technical University

Computer Engineering

CSE222-2021-SPRING

Homework-5 Report

Ferdi Sönmez

161044046

Part2-1:

1)INTRODUCTİON

1.1)Problem Definition

Use the chaining technique for hashing by using linked lists (available in the text book) to chain
items on the same table slot.

2) Problem Solution Approach

I used the chaining technique for hashing, using linked lists to chain. The ones containing the same key values are added to the linkedlist structure.

3)Test Case

a) Many values have been added by defining the Linkedhashmap structure.

```
HashTableChain<Integer,String> myh=new HashTableChain<~>();
myh.put(1,"Ferdi");
myh.put(2,"Ali");
myh.put(3,"Osman2");
myh.put(4,"Osman1");
myh.put(4,"Osman4");
```

b) The items in this structure and wanted to be deleted have been deleted.

```
HashTableChain<Integer,String> myh=new HashTableChain<~>();
myh.put(1,"Ferdi");
myh.put(2,"Ali");
myh.put(3,"Osman2");
myh.put(4,"Osman1");
myh.put(4,"Osman4");
myh.remove( key: 1);
```

c) Returns the number of elements in the structure

```
HashTableChain<Integer,String> myh=new HashTableChain<~>();
myh.put(1,"Ferdi");
myh.put(2,"Ali");
myh.put(3,"Osman2");
myh.put(4,"Osman1");
myh.put(4,"Osman4");
myh.remove( key: 1);
System.out.println("Size:"+myh.size());
```

d) All items in the structure are printed on the screen.

```
Key:2 Value:Ali
Key:3 Value:Osman2
Key:4 Value:Osman1
Key:4 Value:Osman4
```

Part2-2:

1.1)Problem Definition

Use the chaining technique for hashing by using TreeSet (instead of linked list) to chain items on the same table slot.

2) Problem Solution Approach

I used the chaining technique for hashing, TreeSet

to chain. The ones containing the same key values are added to the TreeSet structure.

3)Test Case

a) Many values have been added by defining the TreeSethashmap structure.

```
HashTableChainTree<Integer,String> hashTree=new HashTableChainTree<~>();
hashTree.put(1,"Ali");
hashTree.put(2,"Veli");
hashTree.put(3,"Osman");
hashTree.put(4,"Yakup");
hashTree.put(4,"Unal");
```

b) The items in this structure and wanted to be deleted have been deleted.

```
HashTableChainTree<Integer,String> hashTree=new HashTableChainTree<~>();
hashTree.put(1,"Ali");
hashTree.put(2,"Veli");
hashTree.put(3,"Osman");
hashTree.put(4,"Yakup");
hashTree.put(4,"Unal");
hashTree.remove( key: 2);
```

c) Returns the number of elements in the structure.

```
HashTableChainTree<Integer,String> hashTree=new HashTableChainTree<~>();
hashTree.put(1,"Ali");
hashTree.put(2,"Veli");
hashTree.put(3,"Osman");
hashTree.put(4,"Yakup");
hashTree.put(4,"Unal");
hashTree.remove( key: 2);
hashTree.Print();
System.out.println("Size:"+hashTree.size());
```

d) It is checked whether the structure is full and empty.

```
HashTableChainTree<Integer,String> hashTree=new HashTableChainTree<~>();
hashTree.put(1,"Ali");
hashTree.put(2,"Veli");
hashTree.put(3,"Osman");
hashTree.put(4,"Yakup");
hashTree.put(4,"Unal");
hashTree.remove( key: 2);
hashTree.Print();
System.out.println("Size:"+hashTree.size());
System.out.println("Isempty-->"+hashTree.isEmpty());
```

Part2-3:

1.1)Problem Definition

Use the Coalesced hashing technique. This technique uses the concept of Open Addressing to
find first empty place for colliding element by using the quadratic probing and the concept of
Separate Chaining to link the colliding elements to each other through pointers (indices in the
table). The deletion of a key is performed by linking its next entry to the entry that points the
deleted key by replacing deleted entry by the next entry.

2) Problem Solution Approach

I used the chaining technique for hashing, using linked lists to chain. The % table.length value of the key values is calculated and placed in the appropriate position within the Linkedlist [] structure according to this value.

3)Test Case:

a) Elements have been added to the structure.

```
CoalescedHash<Integer> ch=new CoalescedHash<~>();
ch.put( key: 20);
ch.put( key: 30);
ch.put( key: 40);
ch.put( key: 35);
ch.put( key: 50);
ch.put( key: 60);
ch.put( key: 45);
```

b) Elements have been deleted to the structure.

```
CoalescedHash<Integer> ch=new CoalescedHash<~>();
ch.put( key: 20);
ch.put( key: 30);
ch.put( key: 40);
ch.put( key: 35);
ch.put( key: 50);
ch.put( key: 60);
ch.put( key: 45);
ch.remove( key: 40);
```

c) All the elements in the structure have been written on the screen.

```
CoalescedHash<Integer> ch=new CoalescedHash<~>();
ch.put( key: 20);
ch.put( key: 30);
ch.put( key: 40);
ch.put( key: 35);
ch.put( key: 50);
ch.put( key: 60);
ch.put( key: 45);
ch.remove( key: 40);
ch.Print();
```

d) The number of elements in the structure has been shown.

```
CoalescedHash<Integer> ch=new CoalescedHash<~>();
ch.put( key: 20);
ch.put( key: 30);
ch.put( key: 40);
ch.put( key: 35);
ch.put( key: 50);
ch.put( key: 60);
ch.put( key: 45);
ch.remove( key: 40);
ch.Print();
System.out.println("Size:"+ch.size());
```

e) Whether the building is empty is shown.

```
CoalescedHash<Integer> ch=new CoalescedHash<~>();
ch.put( key: 20);
ch.put( key: 30);
ch.put( key: 40);
ch.put( key: 35);
ch.put( key: 50);
ch.put( key: 60);
ch.put( key: 45);
ch.remove( key: 40);
ch.Print();
System.out.println("Size:"+ch.size());
System.out.println("IsEmpty-->"+ch.isEmpty());
```