```
/ Assignment 2
// © Rajat Sharma
// Written by: Rajat Sharma(ID- 40196467)
```

Assignment 2

Question 1
You are given a hash table with size 7 and a hashing function h(k) = k%7. Output the
state of the table after inserting the below mentioned values in that specific order when
collisions are handled using:
a. Separate chaining
b. Linear probing
c. Double hashing using a second hash function h’(k) = 5 – (k % 5)
The values to be inserted are 19, 26, 13, 48, 17.

Mod of these values with 7 are 5,5,6,6,3

a. Separate chaining
Iteration1
h(0)={}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19}
h(6)={}

Iteration2
h(0)={}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19,26}
h(6)={}

Iteration3

h(0)={}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19, 26}
h(6)={13}

Iteration4
h(0)={}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19, 26}
h(6)={13, 48}

Iteration5
h(0)={}
h(1)={}
h(2)={}
h(3)={17}
h(4)={}
h(5)={19, 26}
h(6)={13, 48}

b. Linear probing
Iteration1
h(0)={}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19}
h(6)={}

Iteration2
h(0)={}
h(1)={}
h(2)={}
h(3)={}

h(4)={}
h(5)={19}
h(6)={26}

Iteration3
h(0)={13}
h(1)={}
h(2)={}
h(3)={}
h(4)={}
h(5)={19}
h(6)={26}

Iteration4
h(0)={13}
h(1)={48}
h(2)={}
h(3)={}
h(4)={}
h(5)={19}
h(6)={26}

Iteration5
h(0)={13}
h(1)={48}
h(2)={}
h(3)={17}
h(4)={}
h(5)={19}
h(6)={26}

c. Double hashing using a second hash function h’(k) = 5 – (k % 5)
Iteration1
Hash function h(k)=k%7
Value to be inserted: 19
H(19)=19 % 7
H(19)=5
We would put the value 19 at index 5

h(0)={}
h(1)={}

h(2)={}
h(3)={}
h(4)={}
h(5)={19}
h(6)={}

Iteration2
H(26)=26%7
H(26)=5
Since there is collision we would use the second hash function
H2(k)=5-(k mod 5)
H2(26)=5-(26 mod 5)
H2(26)=5-1
H2(26)=4
Index at which value is to be added id(H(26)+H2(26))%7
(5+4)%7=2
Hence value added at index 2

Since 2 is empty we would add 26 to index 2

h(0)={}
h(1)={}
h(2)={26}
h(3)={}
h(4)={}
h(5)={19}
h(6)={}

Iteration3
Next we add 13
H(13)=13%7
H(13)=6
Since Index 6 is empty we would add 13 to index 6

h(0)={}
h(1)={}
h(2)={26}
h(3)={}
h(4)={}
h(5)={19}
h(6)={13}

Iteration4
H(48)=48%7

H(48)=6
Since there is collision we would use the second hash function
H2(k)=5-(k mod 5)
H2(48)=5-(48 mod 5)
H2(48)=5-3
H2(48)=2
Index at which value is to be added id(H(48)+H2(48))%7
(6+2)%7=1
Hence value added at index 1

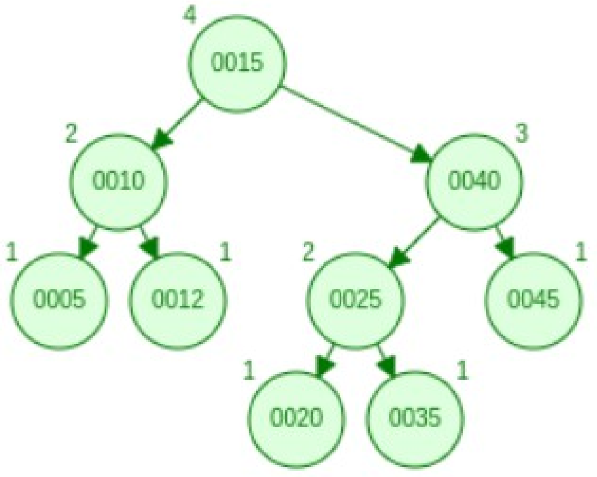
Since 2 is empty we would add 26 to index 2

h(0)={}
h(1)={48}
h(2)={26}
h(3)={}
h(4)={}
h(5)={19}
h(6)={13}

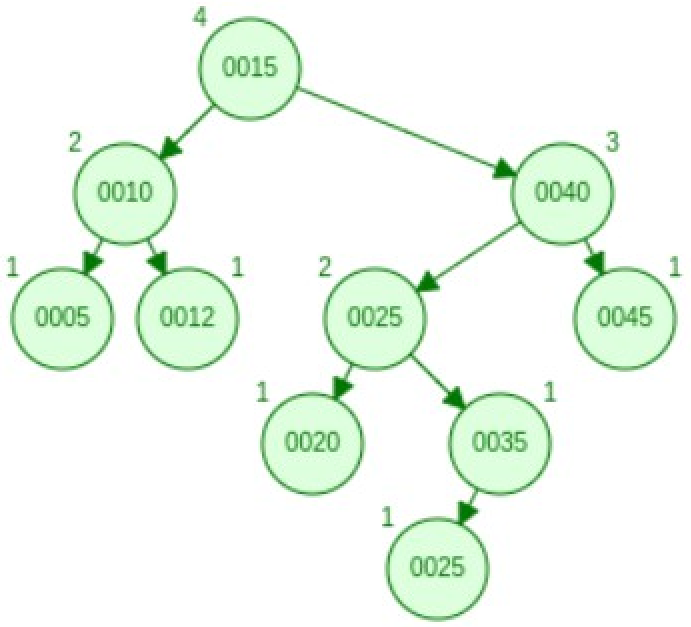
Iteration5
Next we add 17
H(17)=17%7
H(17)=6
Since Index 3 is empty we would add 17 to index 3

h(0)={}
h(1)={48}
h(2)={26}
h(3)={17}
h(4)={}
h(5)={19}
h(6)={13}

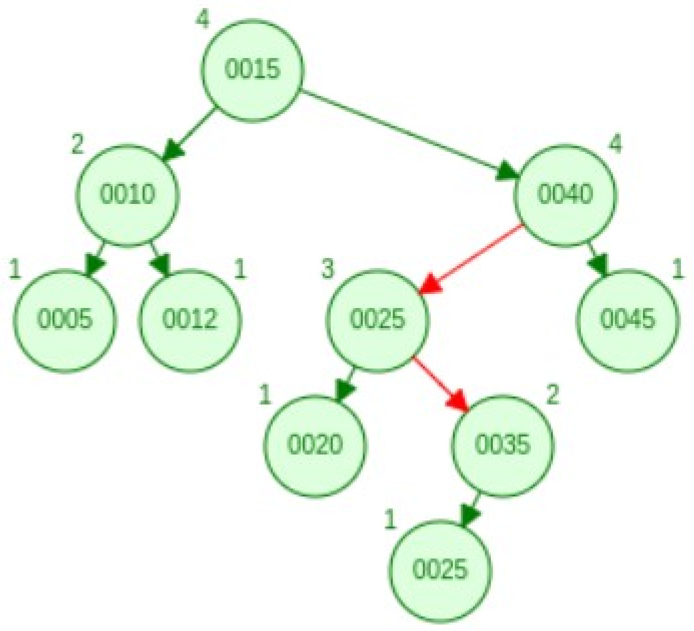
Question 2
Consider the below mentioned AVL tree.
a) Insert key 25 and re-balance the tree if required. Show the progress by drawing each
state of the tree and the final state after the insertion.
b) Remove node with value 40 from the original tree and re-balance the tree if required.
Show the progress by drawing each state of the tree and the final state after the deletion.



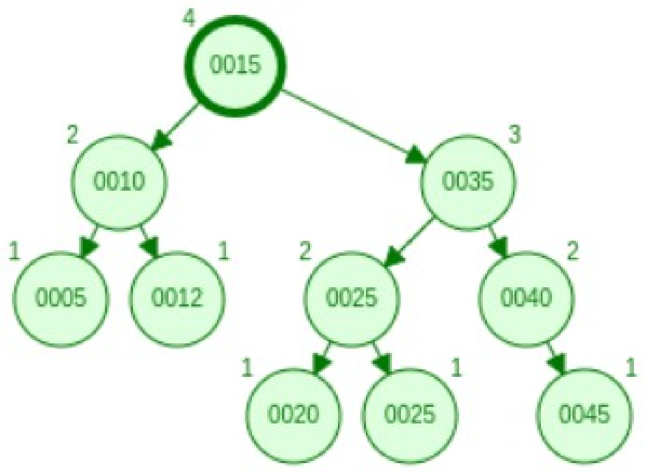
Inserting 25:



After inserting 25, the AVL tree has become unbalanced at node 40 because Right side – left side should always be absolute of less than or equal to 1 but at node 40 the value is 2. So we need to balance it.

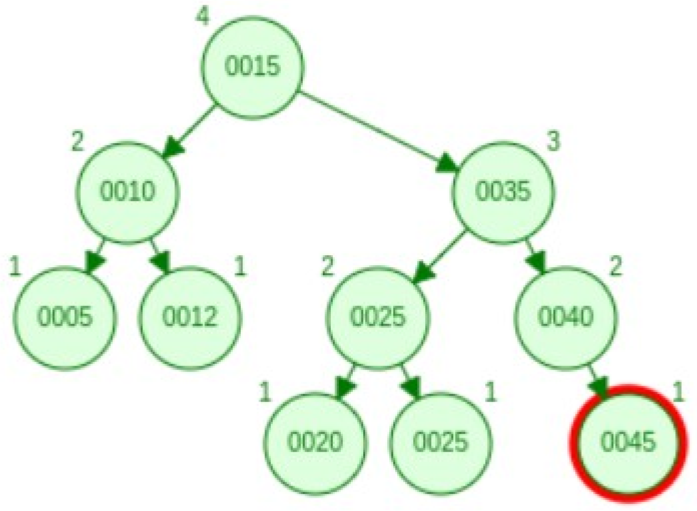


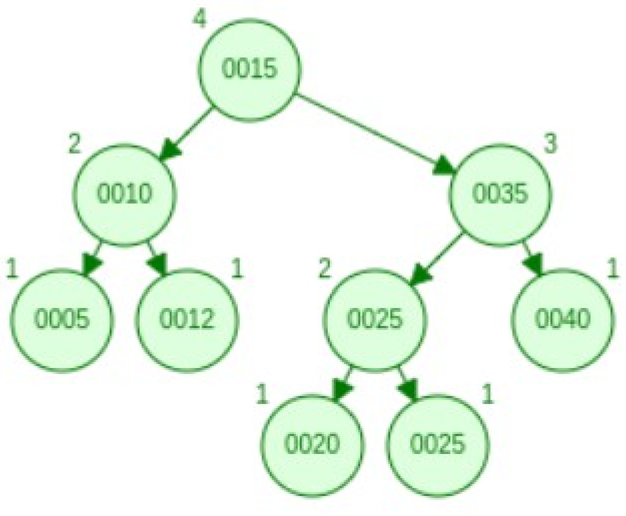
It looks like this after balancing:



Removing: 45-

When we remove 45 all its child nodes become the child of 45’s parent node. The AVL tree looks like this:





Balancing value at each node is less than 2 so the tree is balanced and we don't have to re-balance it again.

Question 3

Develop a well-documented pseudo code to create a method for open addressing which is like double hashing but instead of using a second hash function use a pseudorandom number generator seeded by the key.

a. Will this be more efficient approach compared to double hashing?

b. Will you prefer this over double hashing?

Psuedocode

```
m=7
loop i=0-> arr(len-1)
        check if location arr[i]%m is available,
                put arr[i] in that location
        else
                Random random=seed(Arr[i])
                check if location[arr[i]%m+random%m] is available
                    put arr[i] in that location
        increment the value of i by 1
```

a) This approach will not be more efficient since the seeded values are generated randomly and chances of collision are more.

b)This will be different than double hashing as the seeded values are not random, they are pre decided and used in similar order every-time. Hence double hashing would be preferred over random hashing with psuedorandom function seeded by key.