Ex-1

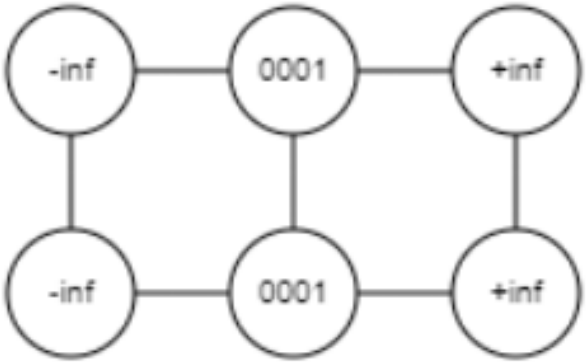
My apologies, I was not able to do analysis part of this question completely due to severe sickness. The code which I have provided uses 3 different Radix sorts according to the type of array i.e 8 bit int or byte, int or 32 bit int , long or 64 bit int. **The rest of the exercise questions are completely done below.**

Ex-2

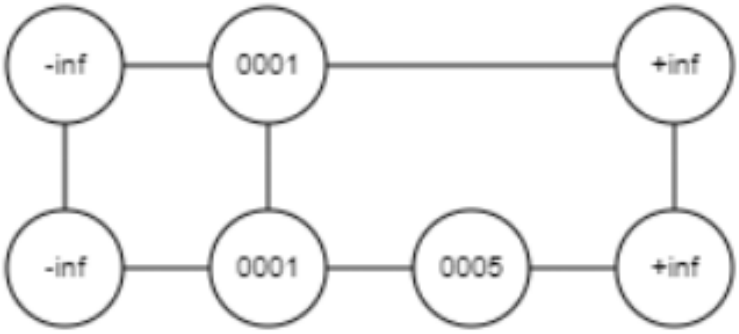
Skip-List Simulation

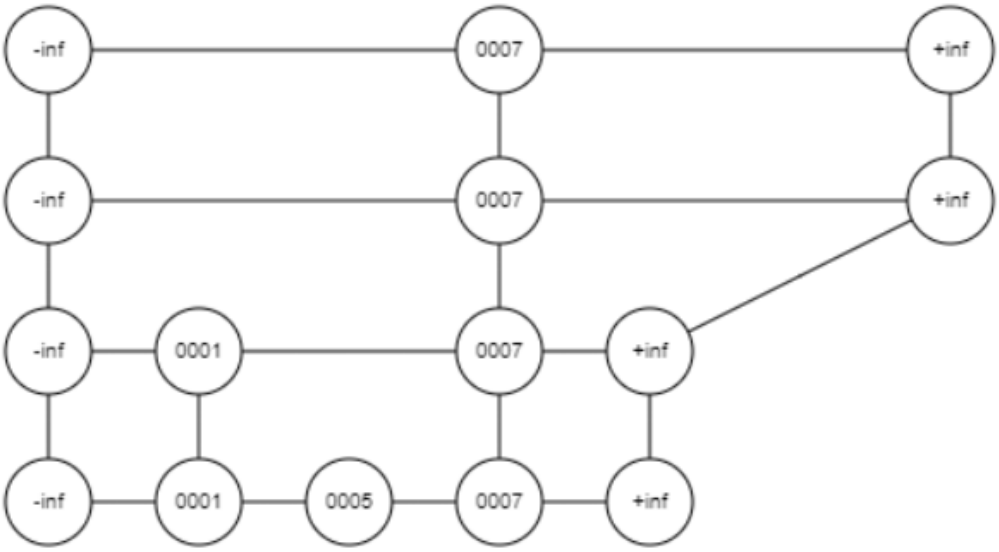
I have chosen element '2' for detail steps of inserting with comments.

Inserting-1: flip coin 2 times 1 heads so one copy on above level.

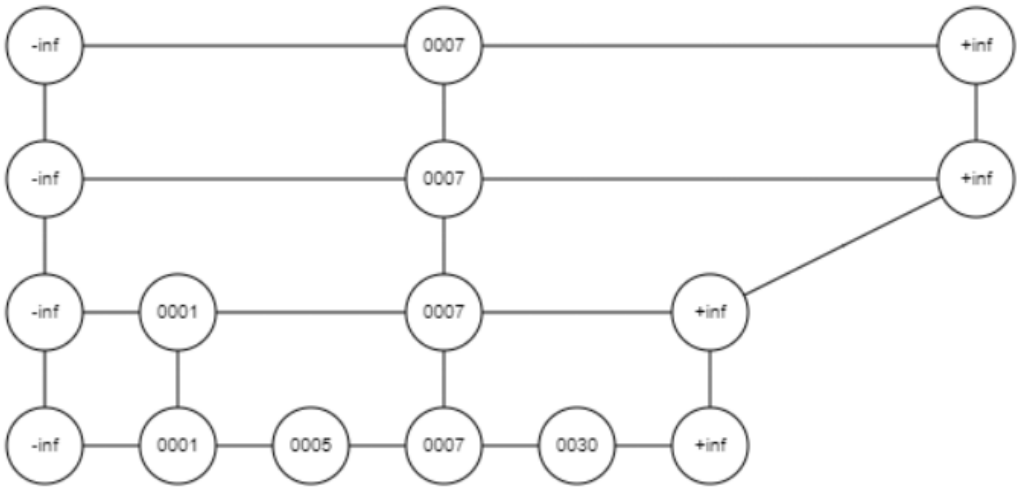


Inserting-5: flip coin 1 times head so no above level.





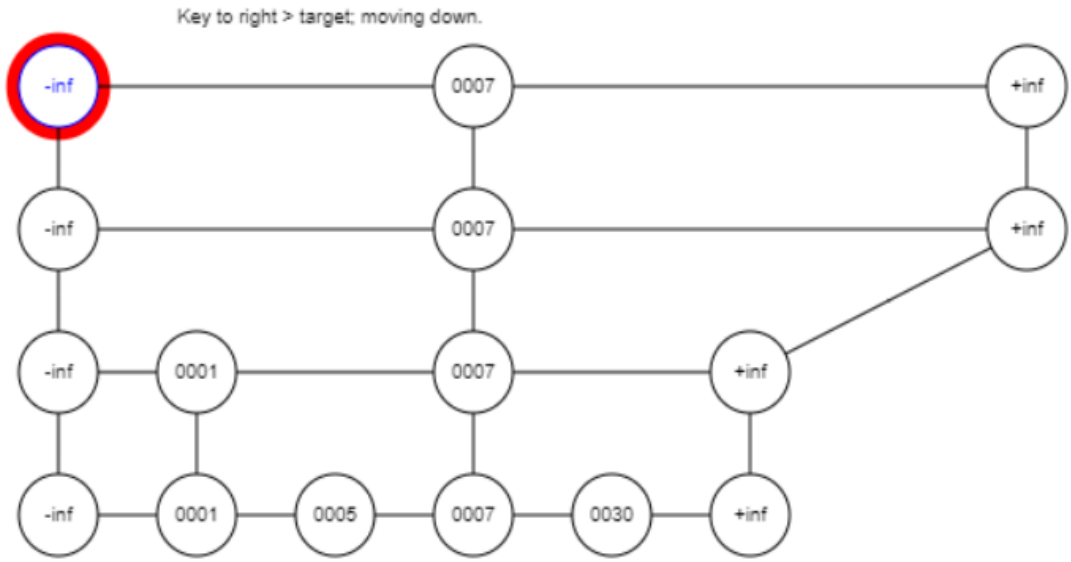
Inserting-7: flip coin 4 times head so 3 copies of 7.



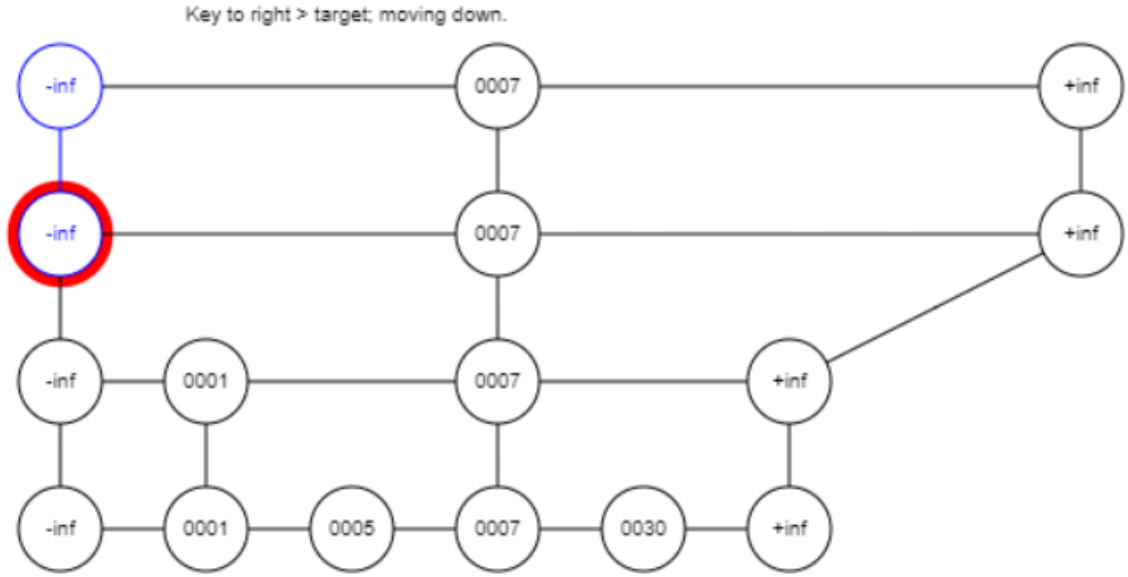
Inserting-30: flip coin 1 times 0 heads so no copies.

Detailed Insertion of ‘2’

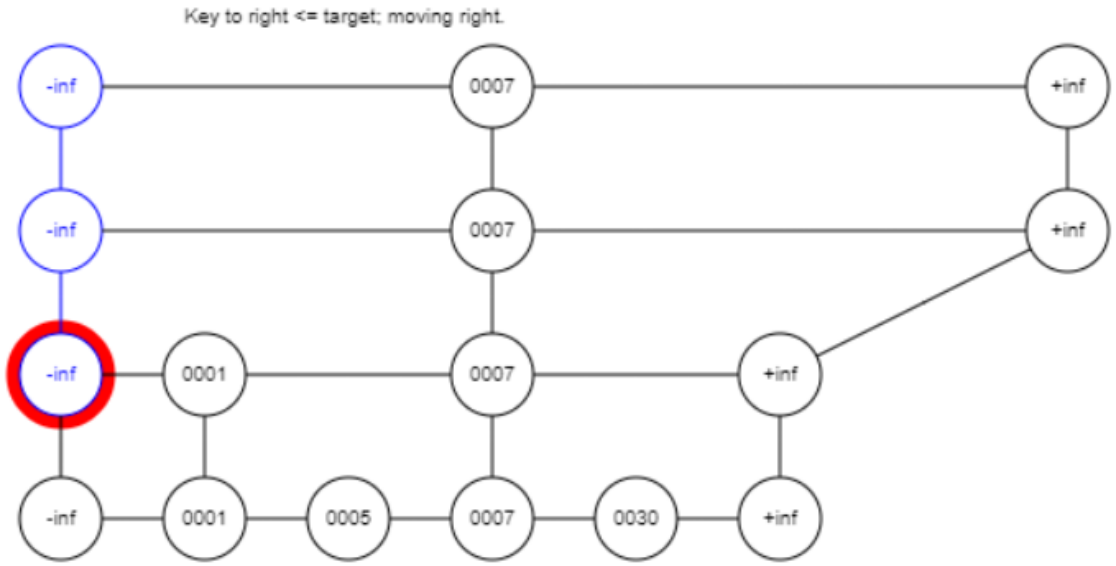
Step-1: As free list is a sorted list the comparisons began at the most higher level starting from -infinity. Here 2> -Infinity and right key is greater than target so it will move to next level from here.



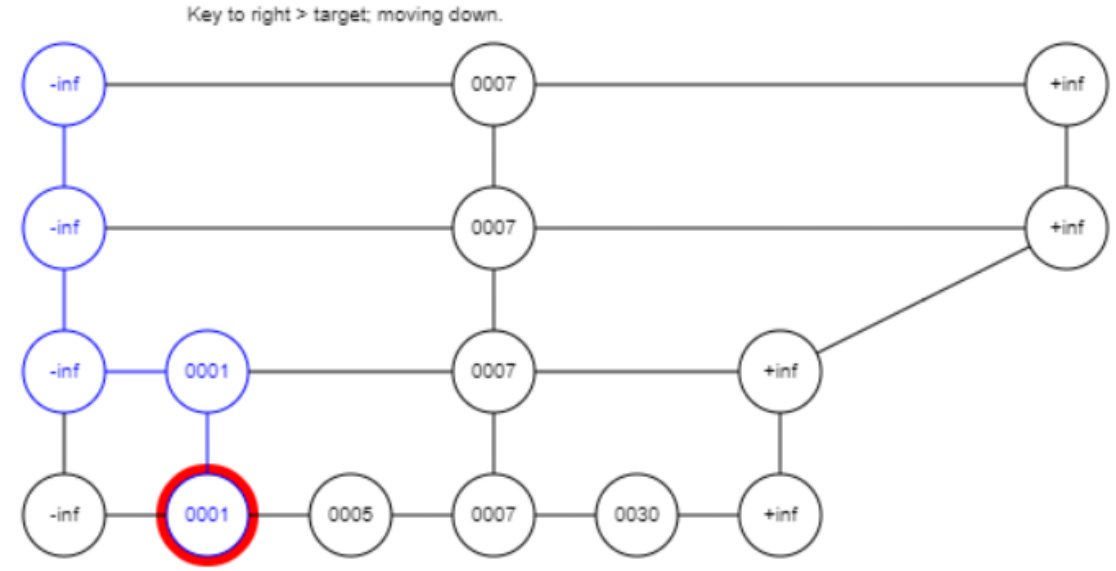
Step-2: This is same as previous case so it will move down to 3rd level as the key to the right is greater than target.



Step-3: Here key to the right element is less than target so the pointer will move right to entry ‘1’.

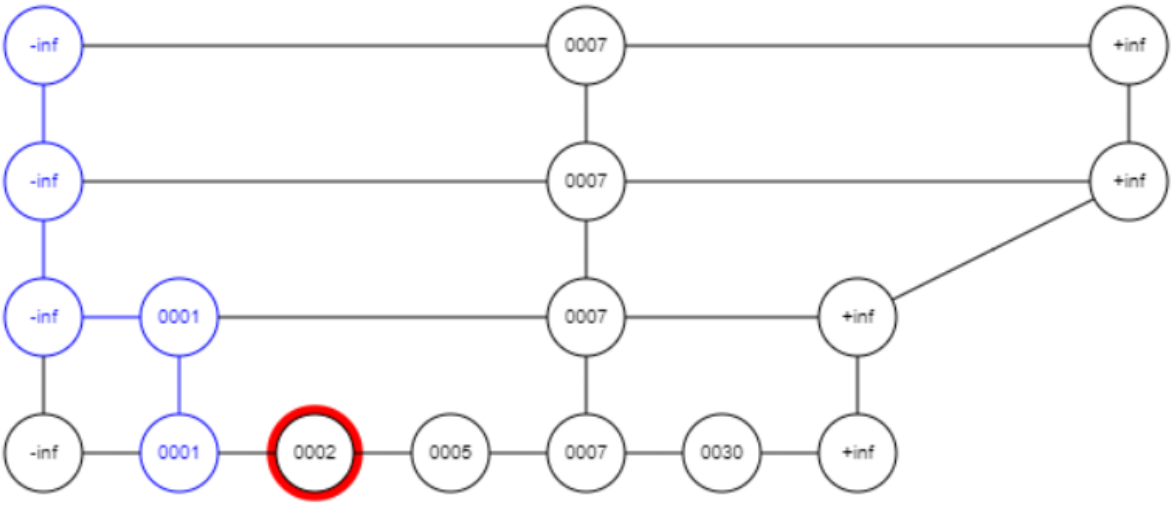


Step-4: Here key to the right element is greater than target so the pointer will

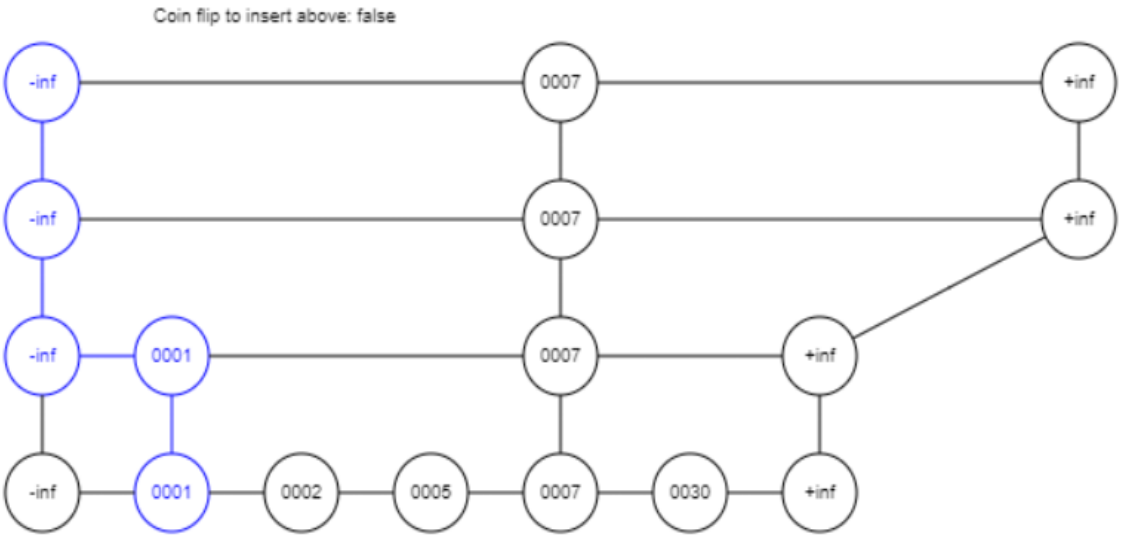


move down to the last level.

Step-5: Here key to the right element is greater than target and there is no further levels downward. So the element will be inserted here.

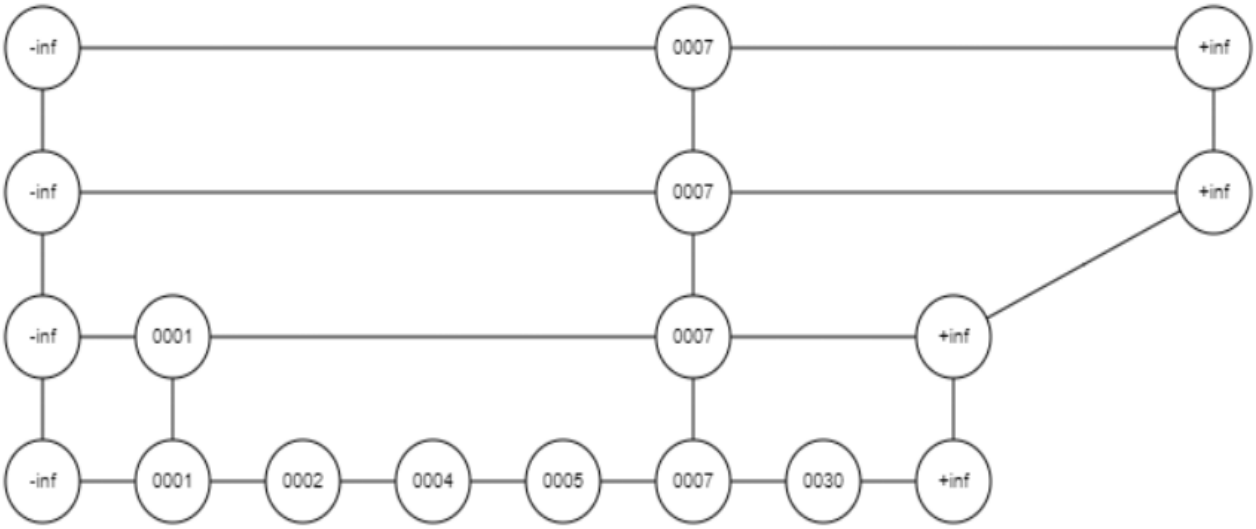


Step-6: Now we will flip the coin in order to check if we want to make a copy of this element in above levels or not.

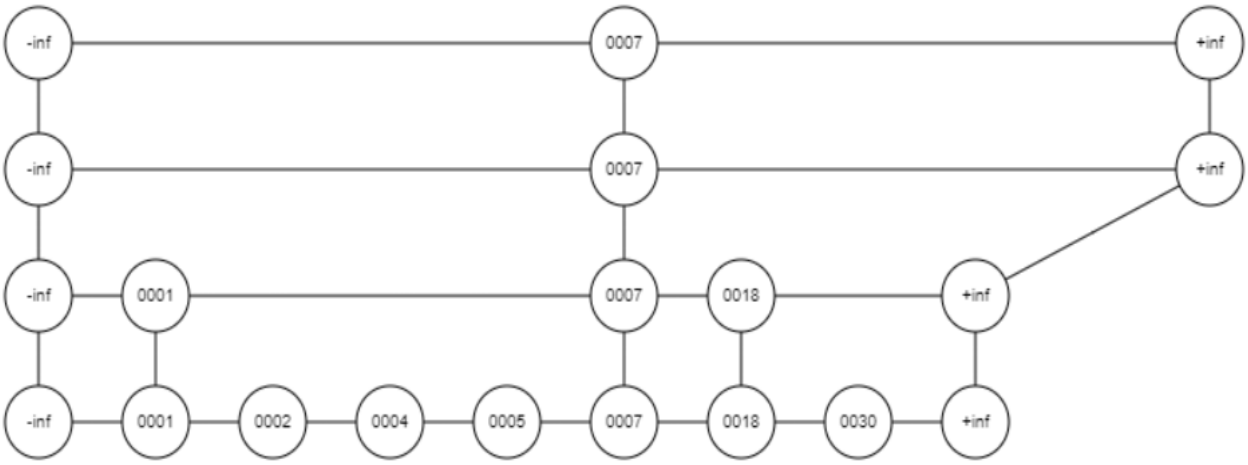


Step-7: As the flip coin is false so we won't create any further copies of thus element in above levels.

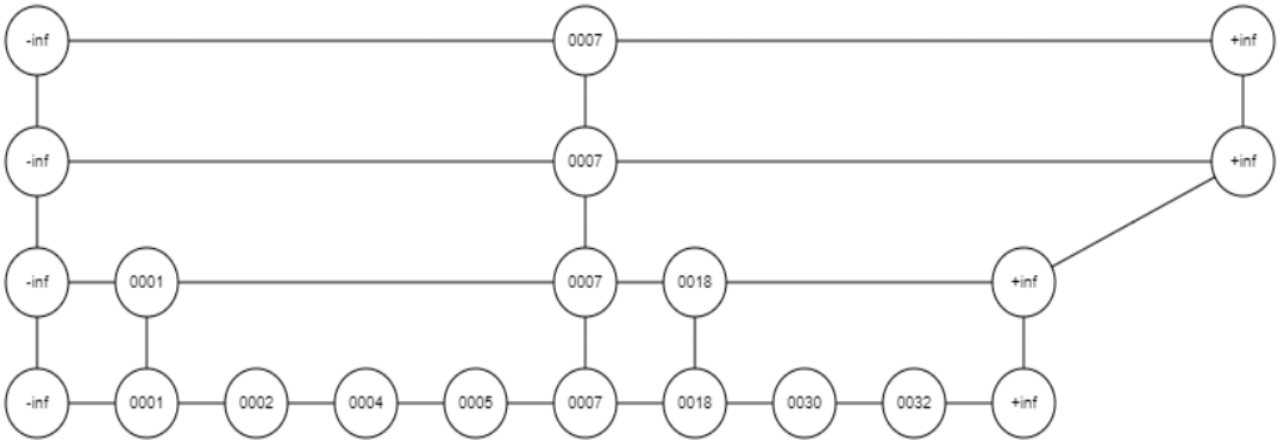
Inserting-4: flip coin 1 times 0 heads so no copy.



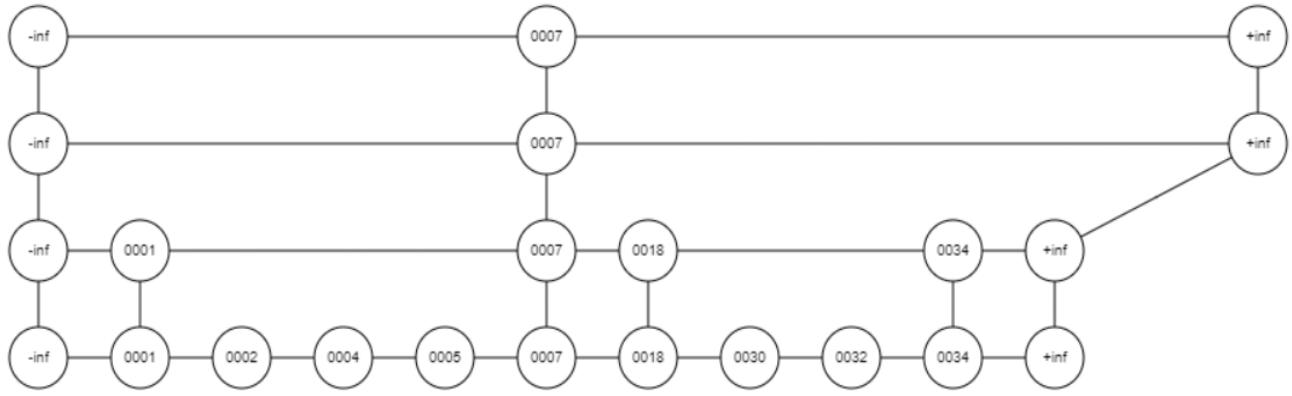
Inserting-18: flip coin 2 times 1 heads so 1 copy of 18 in above level.



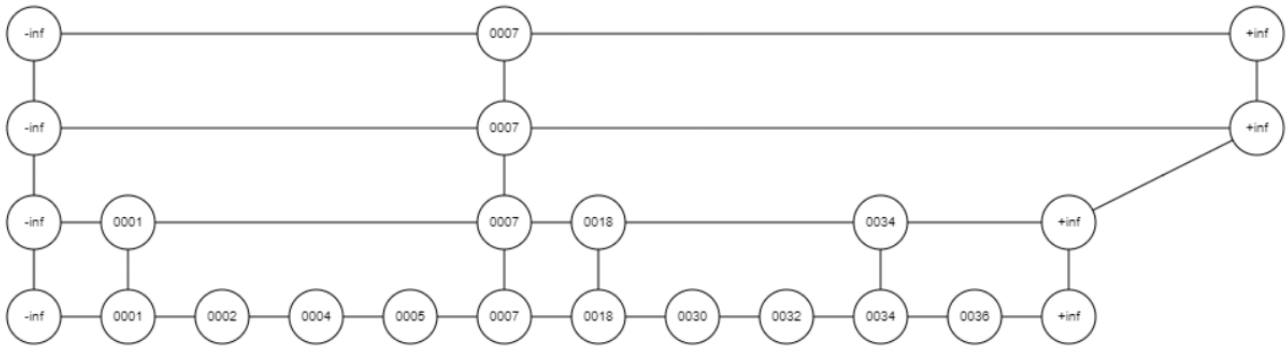
Inserting-32: flip coin 1 times 0 heads so no copy.



Inserting-34: flip coin 2 times 1 heads so 1 copy of 34 in above level.



Inserting-36: flip coin 1 times 0 heads so no copy.

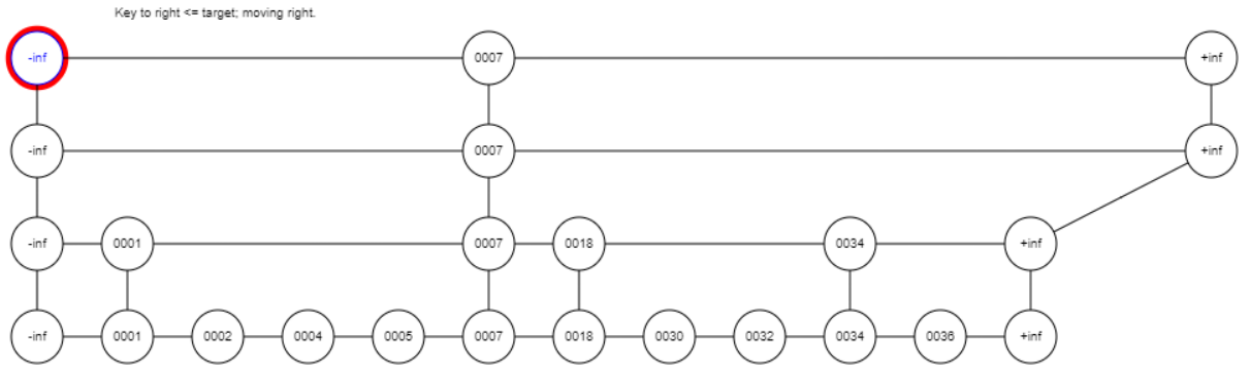


Skip-List Data insertion completed.

Searching for Element-35

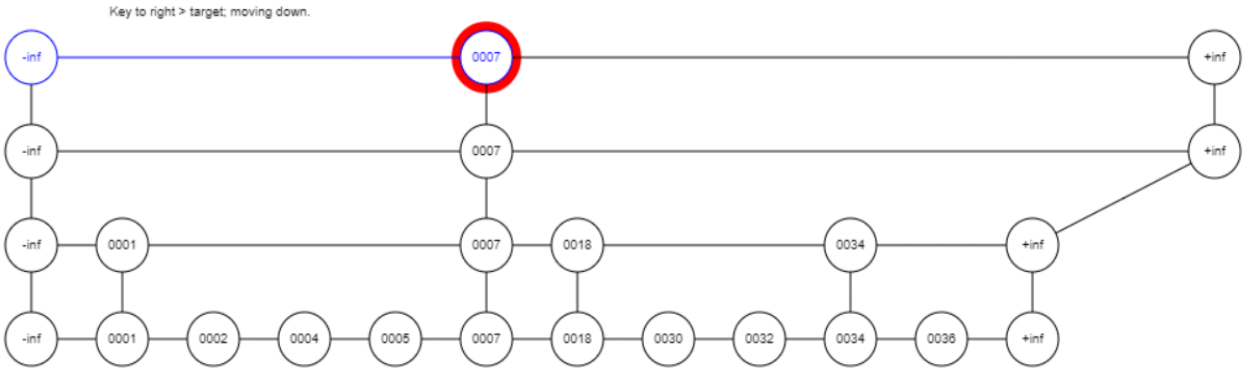
Step-1: We will start from the top as the element on the right side is less than the target so we will move right.

Comparison count=1;



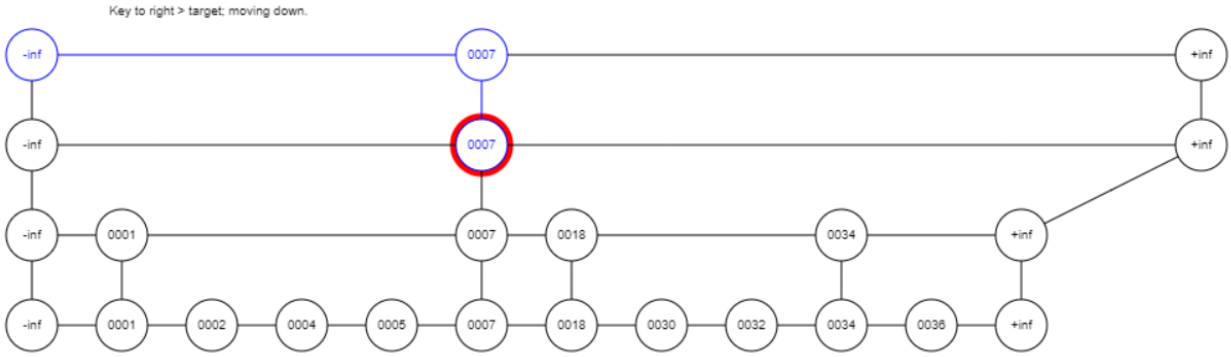
Step-2: Here the right element is +infinity so move down to the next level.

Comparison count=2;



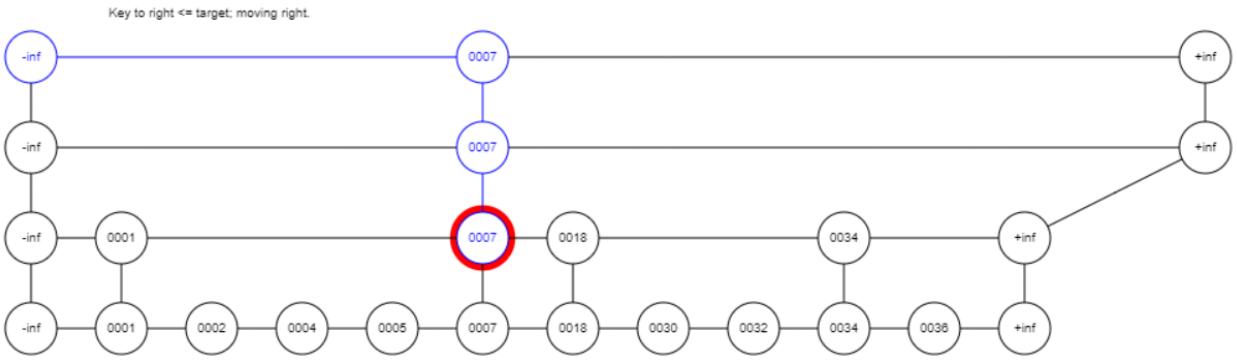
Step-3: Here the right element is +infinity so move down to the next level.

Comparison count=3;



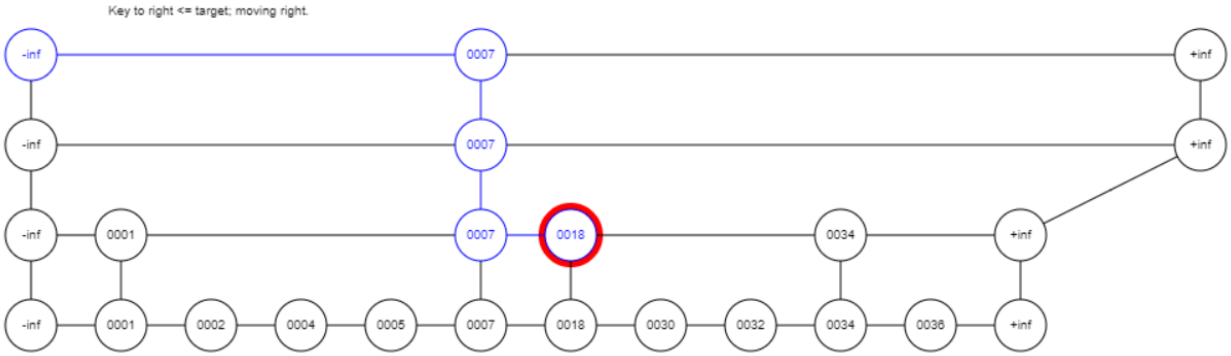
Step-4: Here the right element is +less than the target so we move right.

Comparison count=4;



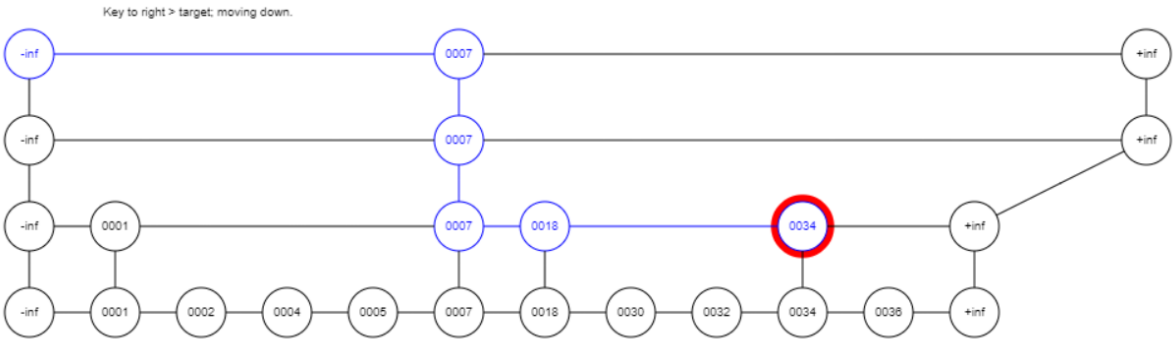
Step-5: Here the right element is less than the target so we move right.

Comparison count=5;



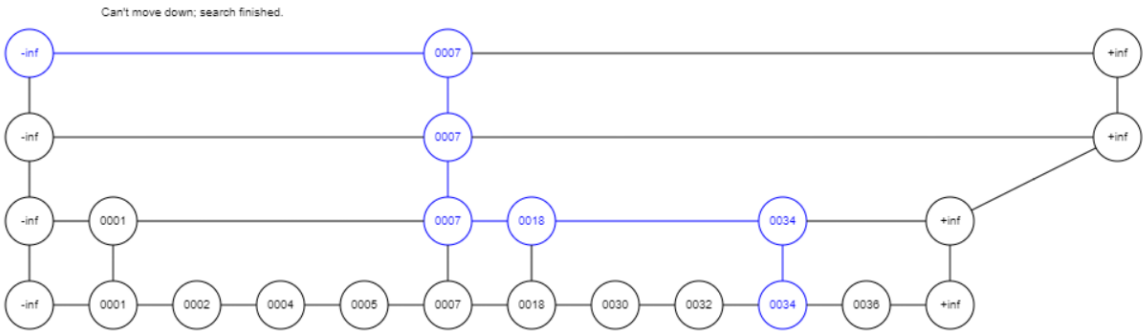
Step-6: Here the right element is +infinity so move down to the next level.

Comparison count=6;



Step-7: Here the right element is greater than the target and there is no level below so the search will terminate here.

Comparison count=7;



Ex-3

Binary Search Simulation.

I have chosen elements ‘7’, ‘2’ and ‘32’ for detailed insertion process.

Inserting-1

Inserting-5: ‘5’ is greater than 1 so it will go right.

Detailed Insertion of 7:

Step-1: Here ‘7’ is greater than ‘1’ so the insertion pointer will move to right child i.e ‘5’

Step-1: Here ‘7’ is greater than ‘5’ so the insertion pointer will move to right child.

Step-1: No further node found so ‘7’ will be inserted here.

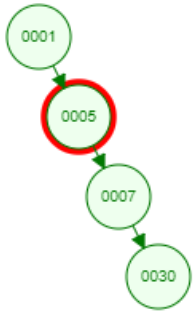
Inserting-30: ‘30’ is greater than all of the current elements so it will go to the right child of last node i.e ‘7’

Detailed Insertion of 2:

Step-1: Here ‘2’ is greater than ‘1’ so the insertion pointer will move to right child i.e ‘5’

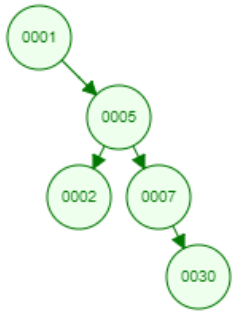
Step-2: Here ‘2’ is less than ‘5’ so the insertion pointer will move to right child of ‘5’.

0002 < 0005. Looking at left subtree

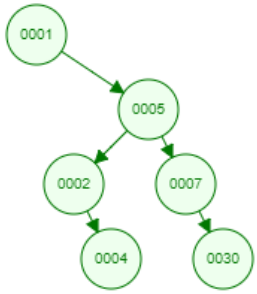


Step-3: No further node found so ‘2’ will be inserted here.

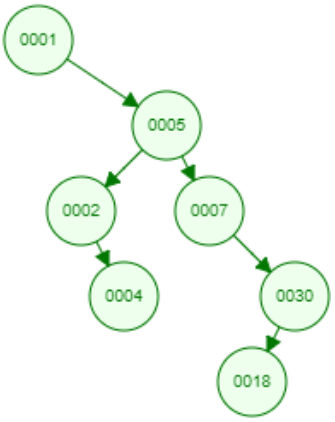
Found null tree, inserting element



Inserting-4: ‘4’ is greater than ‘1’ and is less than ‘5’ and is greater than ‘2’ so ‘4’ will be right child of ‘2’.



Inserting-18: ‘18’ is greater than ‘1’,’5’,’7’ but less than ‘30’ so it will be left child of ‘30’.

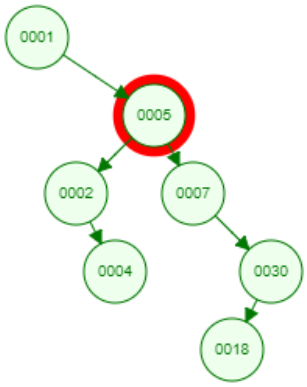


Detailed Insertion of 32:

Step-1: Here ’32’ is greater than ‘1’ so it will move to right child of ‘1’

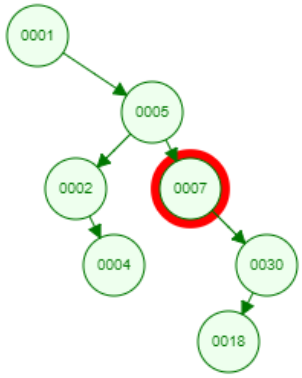
Step-2: Here ’32’ is greater than ‘5’ so it will move to right child of ‘5’

0032 >= 0005. Looking at right subtree



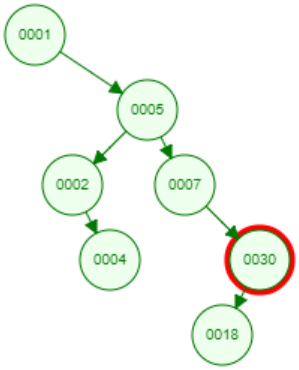
Step-3: Here ’32’ is greater than ‘7’ so it will move to right child of ‘7’

0032 >= 0007. Looking at right subtree



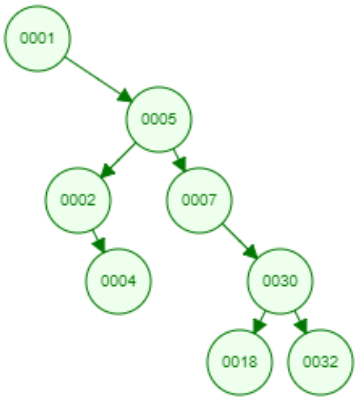
Step-4: Here ’32’ is greater than ‘30’ so it will move to right child of ‘30’

0032 >= 0030. Looking at right subtree

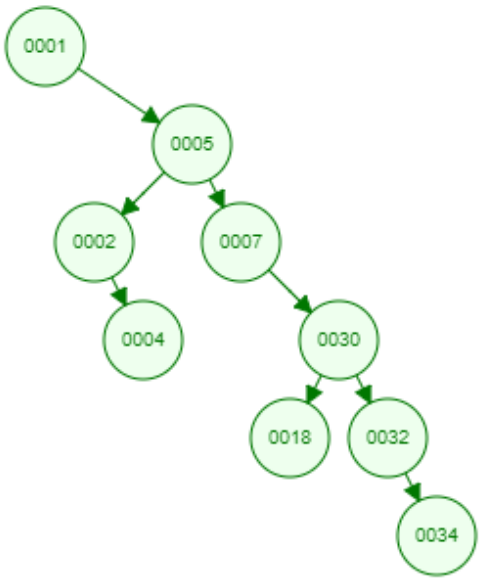


Step-1: No further nodes found so ‘32’ will be inserted as right child of ‘30’

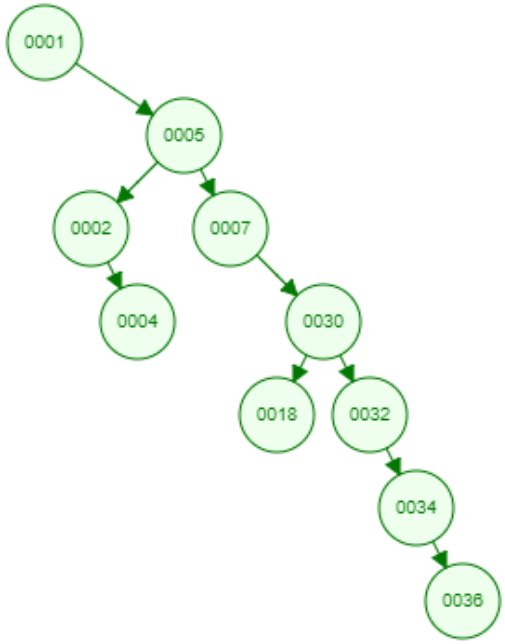
Found null tree, inserting element



Inserting-34: ‘34’ is greater than ‘1’,’5’,’7’,’30’ and ‘32’ So.



Inserting-36: ‘36’ is greater than ‘1’,’5’,’7’,’30’,’32’ and ‘34’ So,



The final BST shows that most of the elements are on the right sub-tree of the root node. This tree is very much right skewed.

Ex-4

AVL Balancing Simulation.

Inserting-1:

Inserting-5:

Step-1: ‘5’ is greater than 1 so it will go on the right hand side of ‘1’

Step-2: Height of tree is adjusted.

Inserting-7:

Step-1: ‘7’ is greater than 1 so it will go on the right hand side of ‘1’

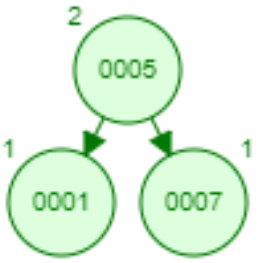
Step-2: ‘7’ is greater than 5 so it will go on the right hand side of ‘5’

Step-3: ‘7’ is inserted and the height of the tree is adjusted

Step-4: As the left – right height of tree exceeds the threshold value ‘-2’

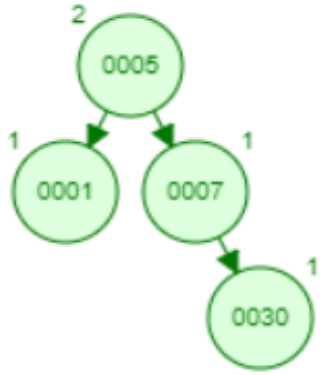
Here it will be a case of single left rotation at node ‘1’.

Step-5: Rotation done and Heights are adjusted.



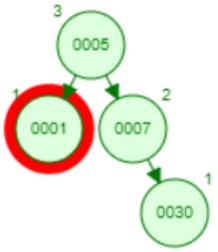
Inserting-30:

Step-1: ‘30’ is greater than ‘5’ and ‘7’ so it will go on the right hand side of ‘7’ and heights are adjusted.



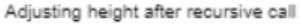
Inserting-2:

Step-1: ‘2’ is less than ‘5’ and is greater than ‘1’ so it will go to the right side of ‘1’ and heights are adjusted.



Inserting-4:

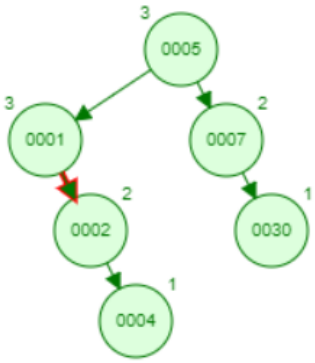
Step-1: ‘4’ is less than ‘5’ and is greater than ‘1’ and ‘2’ so it will go to the right side of ‘2’ and heights are adjusted.



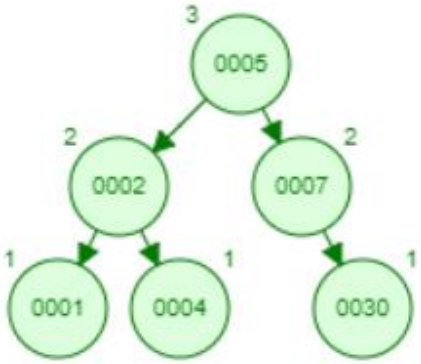
Step-2: As the left – right height of tree exceeds the threshold value ‘-2’

Here it will be a case of single left rotation at node ‘1’.

Single Rotate Left



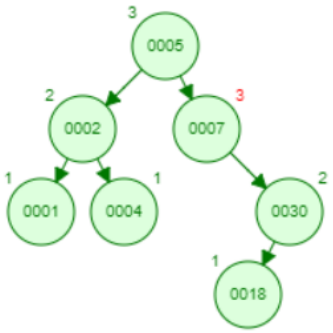
Step-5: Rotation done and Heights are adjusted.



Inserting18

Step1: ‘18’ is greater than ‘5’ and ‘7’ but less than ‘30’ so it will go on the left hand side of ‘30’ and heights are adjusted.

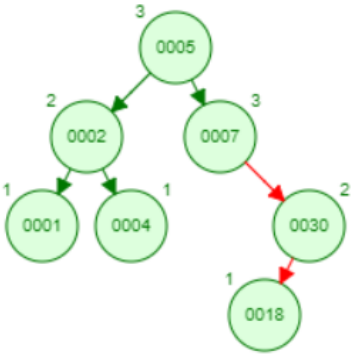
Adjusting height after recursive call



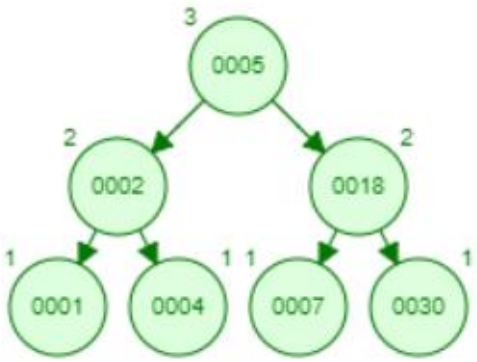
Step2: As the left – right height of tree exceeds the threshold value '-2'

Here it will be a case of Right left Rotation.

Double Rotate Left

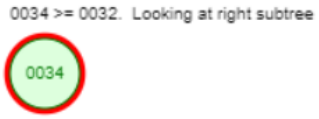


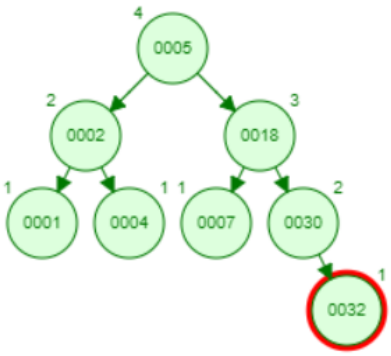
Step3: Rotations Complete Height Adjusted



Inserting 32:

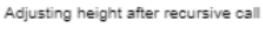
‘32’ is greater than ‘5’, ‘18’,’30’ it will go to right hand side of ‘30’ and heights are adjusted.

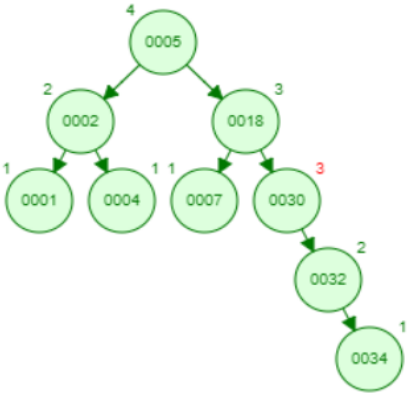




Inserting 34:

Step1: ‘34’ is greater than ‘5’, ‘18’,’30’ and ‘32’ it will go to right hand side of ‘30’ and heights are adjusted.

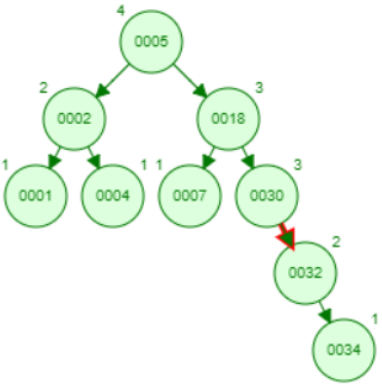




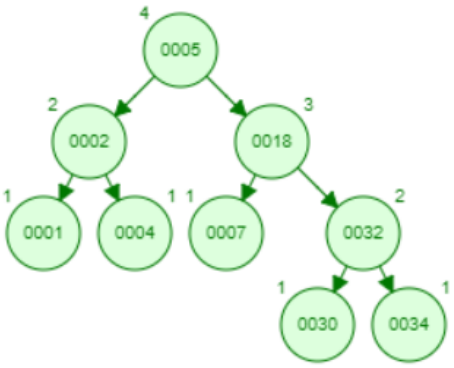
Step2: As the left – right height of tree exceeds the threshold value ‘-2’

Here it will be a case of single left Rotation.

Single Rotate Left



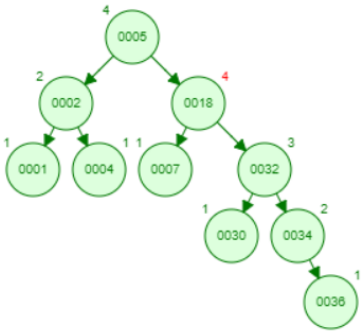
Step3: Rotations Complete Height Adjusted



Inserting 36:

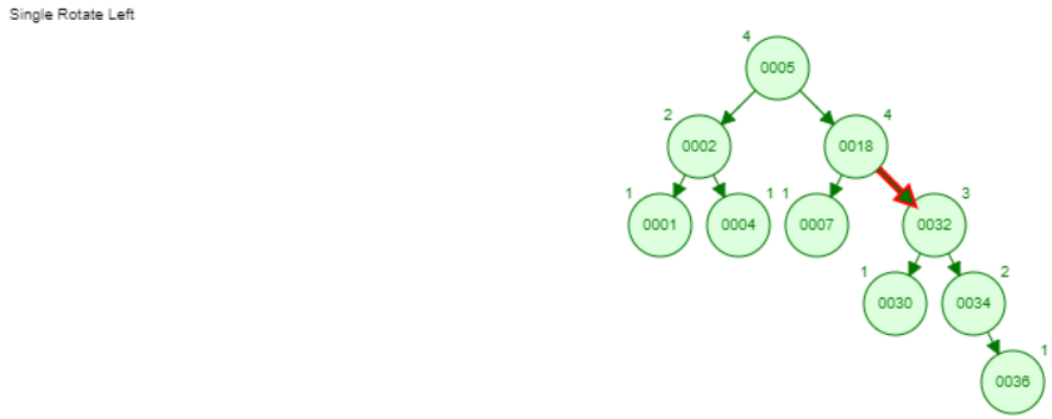
Step1: ‘36’ is greater than ‘5’, ‘18’,’32’ and ‘34’ it will go to right hand side of ‘34’ and heights are adjusted.

Adjusting height after recursive call

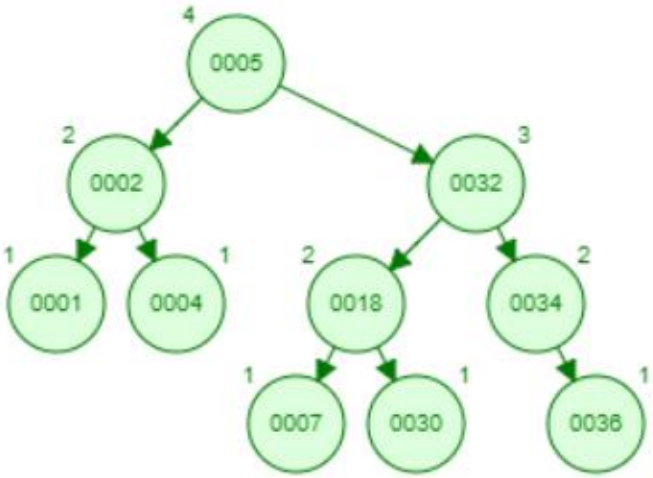


Step2: As the left – right height of tree exceeds the threshold value ‘-2’

Here it will be a case of single left Rotation.



Step3: Rotations Complete Height Adjusted



Total number of Rotations = 6.

Total number of Comparisons for searching ‘35’ = 4.

Ex-5

Red Black Tree Simulation.

Inserting-1:

Inserting-5:

‘5’ is greater than 1 so move right.

Inserting-7:

‘7’ is greater than 1 and 5 so move right of 5.

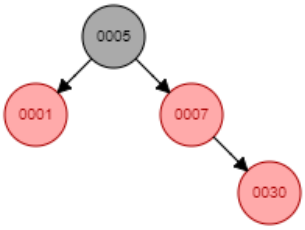
Here we have two red nodes and uncle node is black leaf. So this is case-3. So we Single Rotate Left at Grand Parent and switch the colors of parent and grand-parent.

Single Rotate Left

Inserting-30:

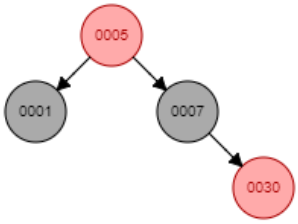
‘30’ is greater than 5,7 so move right of ‘7’.

Found null tree (or phantom leaf), inserting element

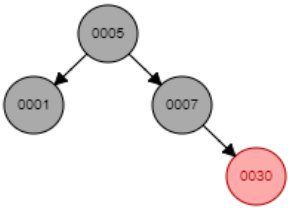


Here we have two red nodes and uncle node is red node So this is case-1. So we switch the colors of parent, grand-parent and uncle.

Node and parent are both red. Uncle of node is red -- push blackness down from grandparent



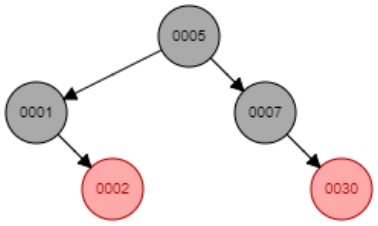
Root must be black so switching colors of Root.



Inserting-2:

‘2 is less than 5 and greater than 1, move right of ‘1’.

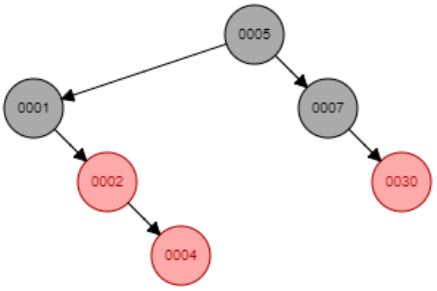
Found null tree (or phantom leaf), inserting element



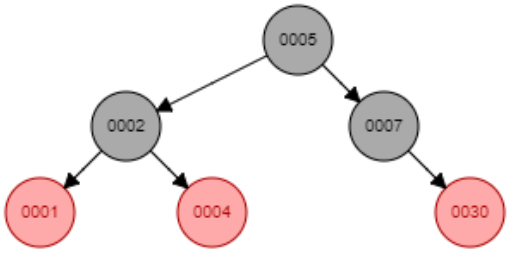
Inserting-4:

‘4 is less than 5 and greater than 1 and 2, move right of ‘2’.

Found null tree (or phantom leaf), inserting element

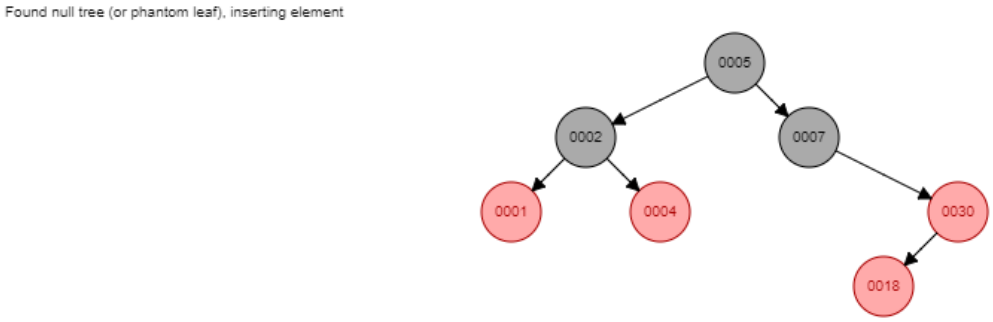


Here we have two red nodes and uncle node is black leaf. So this is case-3. So we Single Rotate Left at Grand Parent and switch the colors of parent and grand-parent.

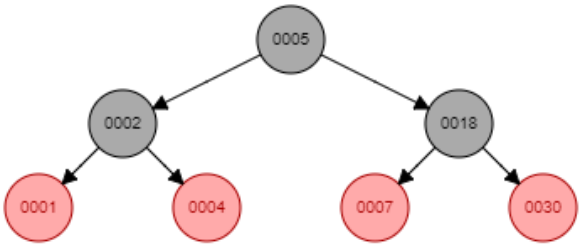


Inserting-18:

‘18 is greater than 5,7 but less than 30, move left of ‘30’.



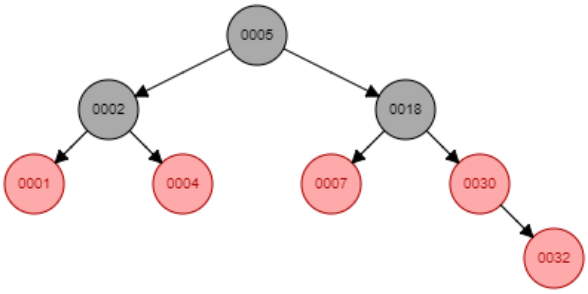
First it is Right Left i.e case-2. We will first right rotate on parent then we will have case-3 i.e two red nodes and uncle node is black leaf.So we Single Rotate Left at Grand Parent and switch the colors of parent and grand-parent.



Inserting-32:

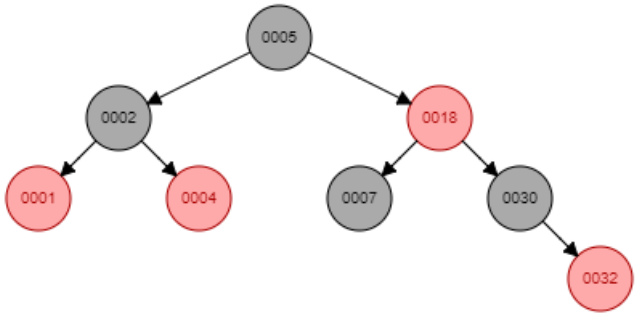
‘32 is greater than 5,18 and 30 move right of ‘30’.

Found null tree (or phantom leaf), inserting element



Here we have two red nodes and uncle node is red node So this is case-1. So we switch the colors of parent, grand-parent and uncle.

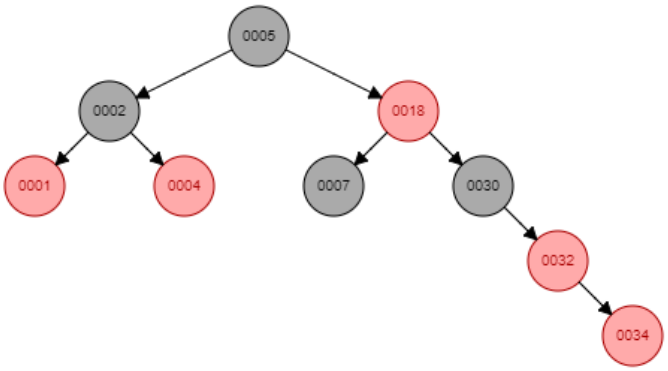
Node and parent are both red. Uncle of node is red -- push blackness down from grandparent



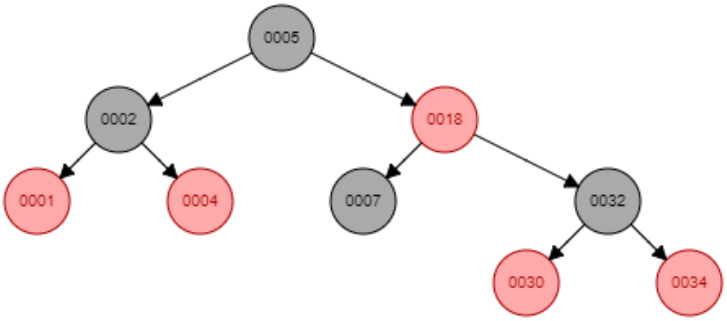
Inserting-34:

‘34 is greater than 5,18 ,30 and 32 move right of ‘32’.

Found null tree (or phantom leaf), inserting element



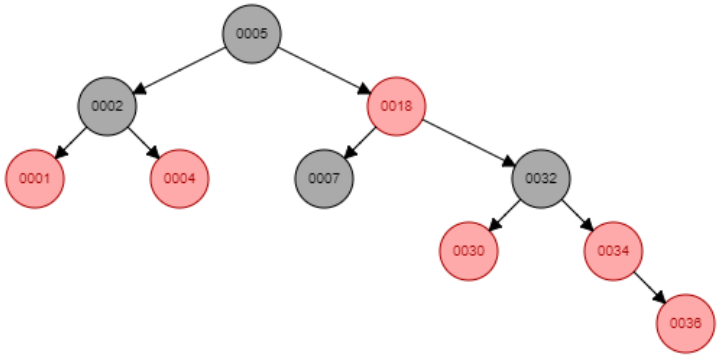
Here we have two red nodes and uncle node is black leaf. So this is case-3. So we Single Rotate Left at Grand Parent and switch the colors of parent and grand-parent.



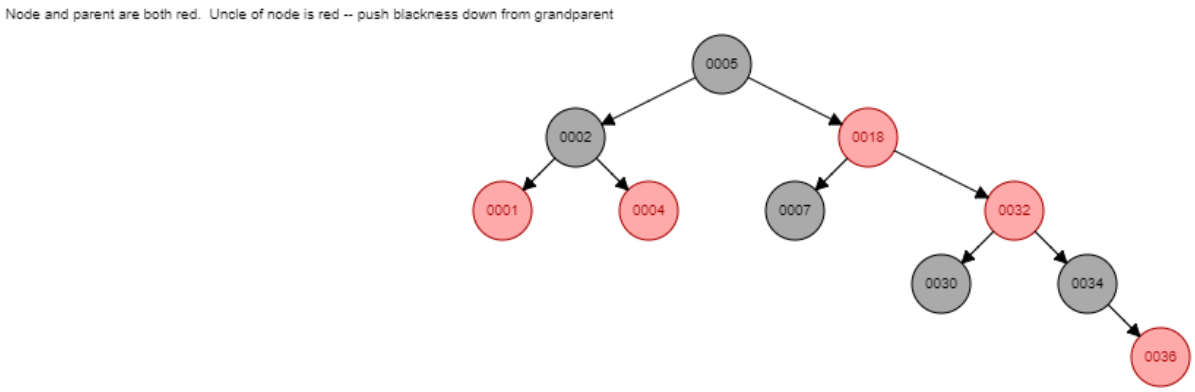
Inserting-36:

‘36 is greater than 5,18 ,32 and 34 move right of ‘34’.

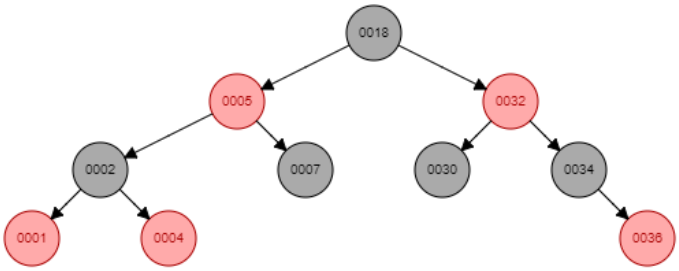
Found null tree (or phantom leaf), inserting element



Here we have two red nodes and uncle node is red node So this is case-1. So we switch the colors of parent, grand-parent and uncle.



But here we have two red nodes and uncle node is black leaf. So this is case-3. So we Single Rotate Left at Grand Parent and switch the colors of parent and grand-parent.



Tree Balanced!

Number of Rotations = 6.