Floor and Ceil from a BST

Given a binary search tree and a key(node) value, find the floor and ceil value for that particular key value.

- **Floor Value Node**: Node with the greatest data lesser than or equal to the key value.
- **Ceil Value Node**: Node with the smallest data larger than or equal to the key value.

Example:

```
        8
      /  \
    4     12
  / \   / \
2    6  10   14
```

Key: *11* ***Floor***: *10* ***Ceil***: *12*
Key: *1*   ***Floor***: *-1* ***Ceil***: *2*
Key: *6*   ***Floor***: *6*   ***Ceil***: *6*
Key: *15* ***Floor***: *14* ***Ceil***: *-1*

There are numerous applications where we need to find the floor/ceil value of a key in a binary search tree or sorted array. For example, consider designing a memory management system in which free nodes are arranged in BST. Find the best fit for the input request.

Ceil in Binary Search Tree using Recursion:
To solve the problem follow the below idea:

Imagine we are moving down the tree, and assume we are root node.
The comparison yields three possibilities,

A) *Root data is equal to key. We are done, root data is ceil value.*
B) *Root data < key value, certainly the ceil value can't be in left subtree.*
Proceed to search on right subtree as reduced problem instance.
C) *Root data > key value, the ceil value may be in left subtree.*
We may find a node with is larger data than key value in left subtree,
if not the root itself will be ceil node.

Below is the implementation of the above approach:

C++

```
// C++ Program to find ceil of a given value in BST

#include <bits/stdc++.h>

using namespace std;


/* A binary tree node has key, left child and right child */

class node {

public:

    int key;

    node* left;

    node* right;

};


/* Helper function that allocates a new node with the given

key and NULL left and right pointers.*/

node* newNode(int key)

{

    node* Node = new node();

    Node->key = key;

    Node->left = NULL;

    Node->right = NULL;

    return (Node);

}


// Function to find ceil of a given input in BST. If input

// is more than the max key in BST, return -1

int Ceil(node* root, int input)

{
    // Base case
    if (root == NULL)
        return -1;

    // We found equal key
    if (root->key == input)
        return root->key;

    // If root's key is smaller, ceil must be in right
    // subtree
    if (root->key < input)
        return Ceil(root->right, input);

    // Else, either left subtree or root has the ceil value
    int ceil = Ceil(root->left, input);
    return (ceil >= input) ? ceil : root->key;
}

// Driver code
int main()
{
    node* root = newNode(8);

    root->left = newNode(4);
    root->right = newNode(12);

    root->left->left = newNode(2);
    root->left->right = newNode(6);


    root->right->left = newNode(10);

    root->right->right = newNode(14);


    for (int i = 0; i < 16; i++)

        cout << i << " " << Ceil(root, i) << endl;


    return 0;

}
```

Output

```
0 2

1 2

2 2

3 4

4 4

5 6

6 6

7 8

8 8

9 10

10 10

11 12

12 12

13 14

14 14

15 -1
```

Time complexity: O(log N)
Auxiliary Space: O(log N)

Below is the implementation to find the floor value:

C++

```
// C++ Program to find floor of a given value in BST

#include <bits/stdc++.h>

using namespace std;


/* A binary tree node has key, left child and right child */

class node {

public:

    int key;

    node* left;

    node* right;

};


/* Helper function that allocates a new node with the given

key and NULL left and right pointers.*/

node* newNode(int key)

{

    node* Node = new node();

    Node->key = key;

    Node->left = NULL;

    Node->right = NULL;

    return (Node);

}
```

```
// Function to find floor of a given input in BST. If input
// is more than the min key in BST, return -1
int Floor(node* root, int input)
{
    // Base case
    if (root == NULL)
        return -1;

    // We found equal key
    if (root->key == input)
        return root->key;

    // If root's key is larger, floor must be in left
    // subtree
    if (root->key > input)
        return Floor(root->left, input);

    // Else, either right subtree or root has the floor
    // value
    else {
        int floor = Floor(root->right, input);
        // exception for -1 because it is being returned in
        // base case
        return (floor <= input && floor != -1) ? floor
                                              : root->key;
    }
}


// Driver code

int main()

{

    node* root = newNode(8);


    root->left = newNode(4);

    root->right = newNode(12);


    root->left->left = newNode(2);

    root->left->right = newNode(6);


    root->right->left = newNode(10);

    root->right->right = newNode(14);


    for (int i = 0; i < 16; i++)

        cout << i << " " << Floor(root, i) << endl;


    return 0;

}
```

Output

```
0 -1

1 -1

2 2

3 2

4 4
```

```
5 4

6 6

7 6

8 8

9 8

10 10

11 10

12 12

13 12

14 14

15 14
```

Time complexity: O(log N)
Auxiliary Space: O(log N)

The iterative approach to find the floor and ceil value in a BST:
To solve the problem follow the below steps:

- If the tree is empty, i.e. root is null, return back to the calling function.
- If the current node address is not null, perform the following steps :
    - If the current node data matches with the key value – We have found both our floor and ceil value.
      Hence, we return back to the calling function.
    - If data in the current node is lesser than the key value – We assign the current node data to the variable keeping track of current floor value and explore the right subtree, as it may contain nodes with values greater than the key value.
    - If data in the current node is greater than the key value – We assign the current node data to the variable keeping track of current ceil value and explore the left subtree, as it may contain nodes with values lesser than the key value.
- Once we reach null, we return back to the calling function, as we have got our required floor and ceil values for the particular key value.

Below is the implementation of the above approach:

C++

```cpp
// C++ program to find floor and ceil of a given key in BST

#include <bits/stdc++.h>

using namespace std;


/* A binary tree node has key, left child and right child */

struct Node {

    int data;

    Node *left, *right;


    Node(int value)

    {

        data = value;

        left = right = NULL;

    }

};


// Helper function to find floor and ceil of a given key in

// BST

void floorCeilBSTHelper(Node* root, int key, int& floor,

                        int& ceil)

{


    while (root) {


        if (root->data == key) {

            ceil = root->data;

            floor = root->data;
            return;
        }

        if (key > root->data) {
            floor = root->data;
            root = root->right;
        }
        else {
            ceil = root->data;
            root = root->left;
        }
    }
    return;
}

// Display the floor and ceil of a given key in BST.
// If key is less than the min key in BST, floor will be -1;
// If key is more than the max key in BST, ceil will be -1;
void floorCeilBST(Node* root, int key)
{

    // Variables 'floor' and 'ceil' are passed by reference
    int floor = -1, ceil = -1;

    floorCeilBSTHelper(root, key, floor, ceil);

    cout << key << ' ' << floor << ' ' << ceil << '\n';
```

```
}


// Driver code
int main()
{
    Node* root = new Node(8);

    root->left = new Node(4);
    root->right = new Node(12);

    root->left->left = new Node(2);
    root->left->right = new Node(6);

    root->right->left = new Node(10);
    root->right->right = new Node(14);

    for (int i = 0; i < 16; i++)
        floorCeilBST(root, i);

    return 0;
}
```

Output

```
0 -1 2

1 -1 2

2 2 2

3 2 4

4 4 4

5 4 6
```

```
6 6 6

7 6 8

8 8 8

9 8 10

10 10 10

11 10 12

12 12 12

13 12 14

14 14 14

15 14 -1
```

Time Complexity: O(log N)
Auxiliary Space: O(1)