Best Code 1 (November)

1./ Function to find vertical sum
#hash map #tree #gfg

```
        1
     /    \
    2      3
  / \     / \
 4   5  6   7
```

The tree has 5 vertical lines

Vertical-Line-1 has only one node 4 => vertical sum is 4Vertical-Line-2: has only one node 2=> vertical sum is 2Vertical-Line-3: has three nodes: 1,5,6 => vertical sum is 1+5+6 = 12Vertical-Line-4: has only one node 3 => vertical sum is 3Vertical-Line-5: has only one node 7 => vertical sum is 7

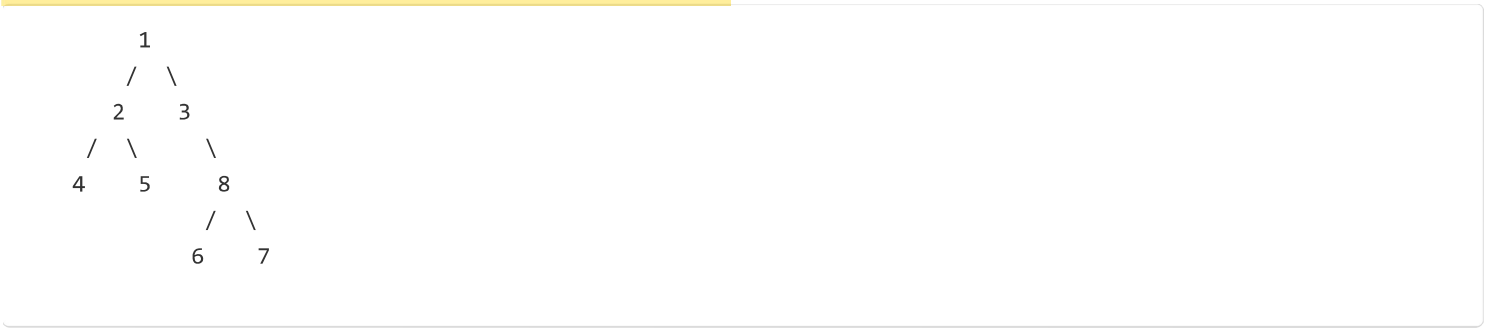
So expected output is 4, 2, 12, 3 and 7

```
void verticalSumUtil(Node *node, int hd,
                map<int, int> &Map)
{
   // Base case
   if (node == NULL) return;

   // Recur for left subtree
   verticalSumUtil(node->left, hd-1, Map);

   // Add val of current node to
   // map entry of corresponding hd
   Map[hd] += node->data;

   // Recur for right subtree
   verticalSumUtil(node->right, hd+1, Map);
}
void verticalSum(Node *root)
{
   // a map to store sum of nodes for each
   // horizontal distance
   map < int, int> Map;
   map < int, int> :: iterator it;
```

```
    // populate the map
    verticalSumUtil(root, 0, Map);

    // Prints the values stored by VerticalSumUtil()
    for (it = Map.begin(); it != Map.end(); ++it)
    {
        cout << it->first << ": "
             << it->second << endl;
    }
}
```

2.

// Function to find the maximum width of the tree
// using level order traversal #queue Basic #tree #gfg



For the above tree, width of level 1 is 1, width of level 2 is 2, width of level 3 is 3 width of level 4 is 2. So the maximum width of the tree is 3.

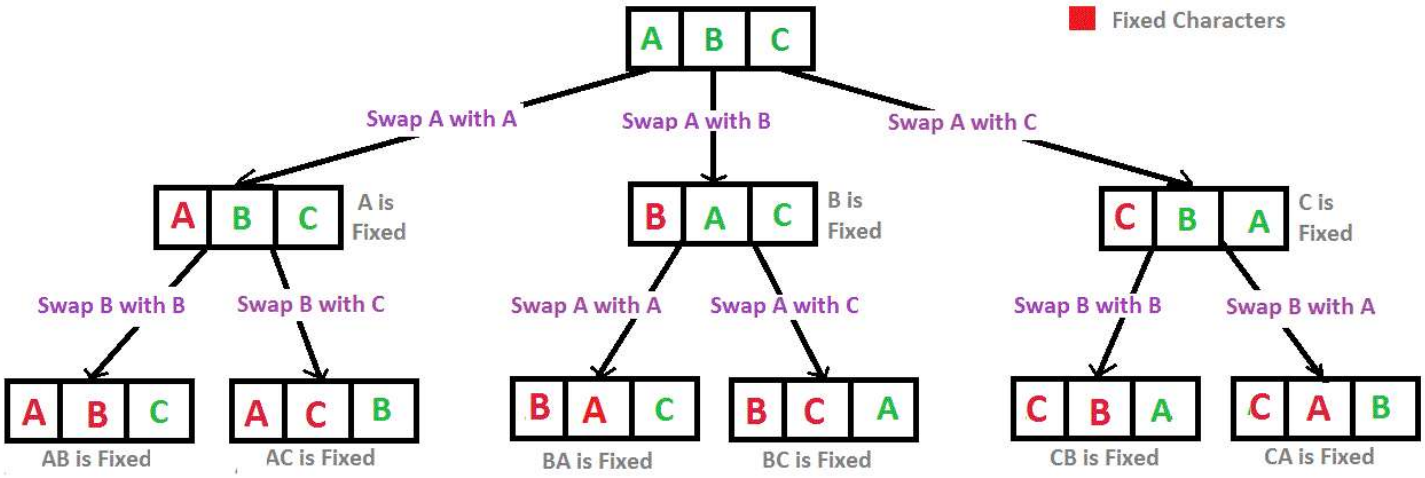
```
int maxWidth(struct Node* root)
{
    // Base case
    if (root == NULL)
        return 0;

    // Initialize result
    int result = 0;

    // Do Level order traversal keeping track of number
    // of nodes at every level.
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        // Get the size of queue when the level order
        // traversal for one level finishes
```

```
        int count = q.size();

        // Update the maximum node count value
        result = max(count, result);

        // Iterate for all the nodes in the queue currently
        while (count--) {
            // Dequeue an node from queue
            Node* temp = q.front();
            q.pop();

            // Enqueue left and right children of
            // dequeued node
            if (temp->left != NULL)
                q.push(temp->left);
            if (temp->right != NULL)
                q.push(temp->right);
        }
    }

    return result;
}
```

3.Write a program to print all permutations of a given string
#backtracking #gfg

```
// Function to print permutations of string
// This function takes three parameters:
// 1. String
// 2. Starting index of the string
// 3. Ending index of the string.
void permute(string a, int l, int r)
{
    // Base case
    if (l == r)
        cout<<a<<endl;
    else
    {
        // Permutations made
        for (int i = l; i <= r; i++)
        {
```

```
        // Swapping done
        swap(a[l], a[i]);

        // Recursion called
        permute(a, l+1, r);

        //backtrack
        swap(a[l], a[i]);
    }
  }
```



Recursion Tree for Permutations of String "ABC"

```
}
```

4.Find the Missing Number
#XOR

```
Input: arr[] = {1, 2, 4, 6, 3, 7, 8}
Output: 5
Explanation: The missing number from 1 to 8 is 5
```

```
// Function to get the missing number
int getMissingNo(int a[], int n)
{
   // For xor of all the elements in array
   int x1 = a[0];
   // For xor of all the elements from 1 to n+1
   int x2 = 1;
   for (int i = 1; i < n; i++)
```

```
    x1 = x1 ^ a[i];
  for (int i = 2; i <= n + 1; i++)
    x2 = x2 ^ i;
  return (x1 ^ x2);
}
```