Binary Tree

116. Populating Next Right Pointers in Each Node

117. Populating Next Right Pointers in Each Node II

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* next;

    Node() : val(0), left(NULL), right(NULL), next(NULL) {}

    Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}

    Node(int _val, Node* _left, Node* _right, Node* _next)
        : val(_val), left(_left), right(_right), next(_next) {}
};
*/
//Mehtod 1: use bfs, it need extra space more than o(n/2), but it can handle non-perferct binary tree

class Solution {
public:
    Node* connect(Node* root) {
        // use queue to bfs;// space Q(one  level ???)
        if (root == NULL) return NULL;
        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            int n = q.size() - 1;
            for (int i = 0; i <= n; i++) {
                Node* node = q.front();
                q.pop();
                if (i == n) node->next = NULL;
                else {
                    Node* nextnode = q.front();
                    node->next = nextnode;
                }
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return root;
    }
};


// space require o(1) so we need use perferct BT condition
//method2: search on tree branch trees
```

```
class Solution {
public:
    Node* connect(Node* root) {

        if (root == NULL) return NULL;
        connectTree(root->left, root->right);
        return root;
    }

    void connectTree(Node* node1, Node* node2) {
        if (node1 == NULL || node2 == NULL)
            return;
        node1->next = node2;
        connectTree(node1->left, node1->right);
        connectTree(node2->left, node2->right);
        connectTree(node1->right, node2->left);

    }
};
```