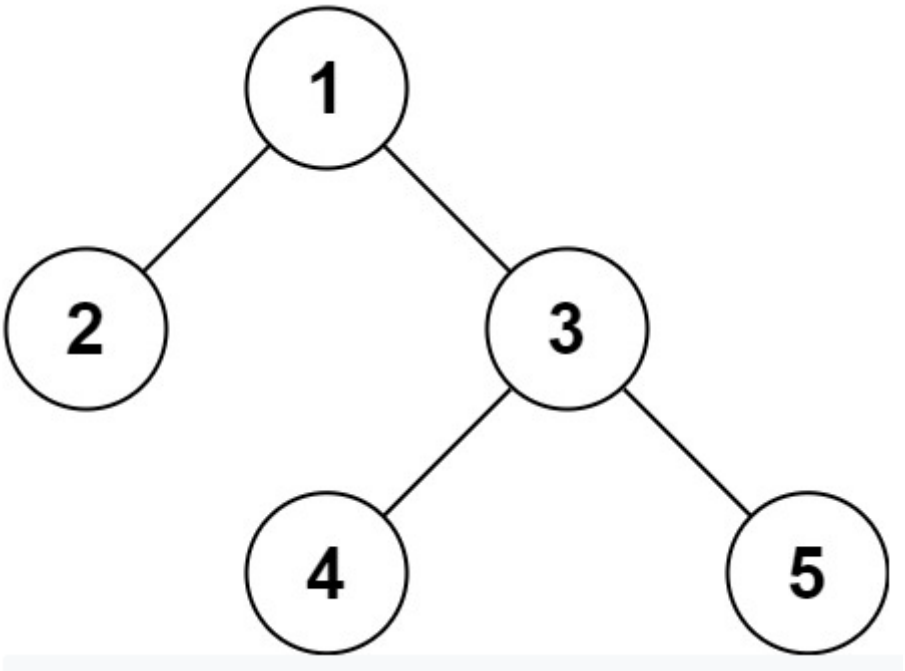
https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Clarification: The input/output format is the same as how LeetCode serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Example 1:



Input: root = [1,2,3,null,null,4,5]

Output: [1,2,3,null,null,4,5]

Example 2:

Input: root = []

Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 10^4].
- -1000 <= Node.val <= 1000

What does [1,null,2,3] mean in binary tree representation?

https://support.leetcode.com/hc/en-us/articles/360011883654-What-does-1-null-2-3-mean-in-binary-tree-representation

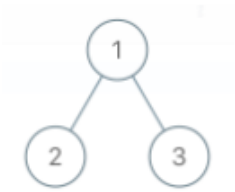
-The input [1,null,2,3] represents the *serialized* format of a binary tree using **level order traversal**, **where null signifies a path terminator where no node exists below**.

Examples:

1. `[]`

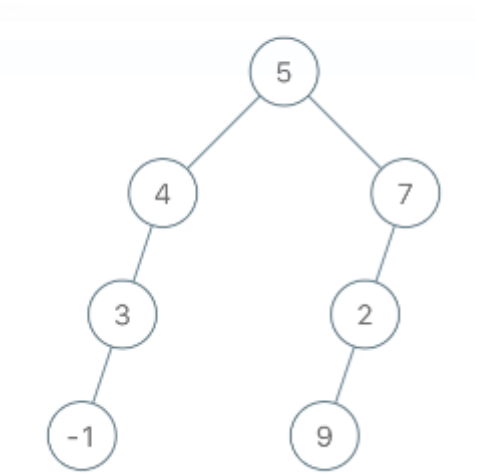
   Empty tree.
   The root is a reference to `NULL` (C/C++), `null` (Java/C#/Javascript), `None` (Python), or `nil` (Ruby).

2. `[1,2,3]`



3. `[1,null,2,3]`

4. `[5,4,7,3,null,2,null,-1,null,9]`



Attempt 1: 2022-11-01

Solution 1: Preorder traversal for Serialize using DFS and for Deserialize using DFS (based on Queue) (30min)

Two key points:

1. Use preorder traversal (root -> left -> right) and mark NULL as "#"

2. Deserialize with Queue based on preorder sequence (root -> left -> right)

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {
    private String NA = "X";
    private String spliter = ",";

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        serializeHelper(root, sb);
        return sb.toString();
    }
    /**
     e.g
                5                                       5
              /   \                                   /     \
             3     6   => if considering NULL(x) =>  3       6
           /  \    \                                / \     /  \
          2    4    7                              2   4    x   7
                                                   / \ / \       / \
                                                  x  x x  x     x   x

```

```java
        preorder serialize into string: 5,3,2,X,X,4,X,X,6,X,7,X,X,
    */
    // Style 1
    private void serializeHelper(TreeNode root, StringBuilder sb) {
        // Base case: Handle NULL
        if(root == null) {
            sb.append(NA).append(spliter);
            return;
        }
        // Preorder traversal
        sb.append(root.val).append(spliter);
        serializeHelper(root.left, sb);
        serializeHelper(root.right, sb);
    }

    // Style 2
    //private void serializeHelper(TreeNode root, StringBuilder sb) {
    //    if(root == null) {
    //        sb.append(NA).append(spliter);
    //    } else {
    //        sb.append(root.val).append(spliter);
    //        serializeHelper(root.left, sb);
    //        serializeHelper(root.right, sb);
    //    }
    //}

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        Queue<String> q = new LinkedList<String>();
        q.addAll(Arrays.asList(data.split(spliter)));
        return buildTree(q);
    }

    // Decode preorder traversal (5,3,2,X,X,4,X,X,6,X,7,X,X,) into tree
    private TreeNode buildTree(Queue<String> q) {
        String rootVal = q.poll();
        if(rootVal.equals(NA)) {
            return null;
        }
        TreeNode root = new TreeNode(Integer.valueOf(rootVal));
```

```
        // Based on preorder, first build left subtree, then right subtree,
        // and on each recursion Queue will pop out one element, since Queue
        // is a object and no backtrack here, the number of elements on
        // Queue will keep decreasing
        root.left = buildTree(q);
        root.right = buildTree(q);
        return root;
    }
}
// Your Codec object will be instantiated and called as such:
// Codec ser = new Codec();
// Codec deser = new Codec();
// TreeNode ans = deser.deserialize(ser.serialize(root));

Time Complexity: O(N), where N <= 10^4 is number of nodes in the Binary Tree.
Space Complexity: O(N)
```

Difference between L297.Serialize and Deserialize Binary Tree (use only preorder to construct tree) and L105.Construct Binary Tree from Preorder and Inorder Traversal (use both preorder and inorder to construct tree) ?

Refer to

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/discuss/74253/Easy-to-understand-Java-Solution/269310

Difference between reconstruct the tree #105 **preorder/postorder + inorder** and this problem which just uses **preorder**

1. #105 preorder/postorder + inorder: **why we have to use 2 lists/traversals**

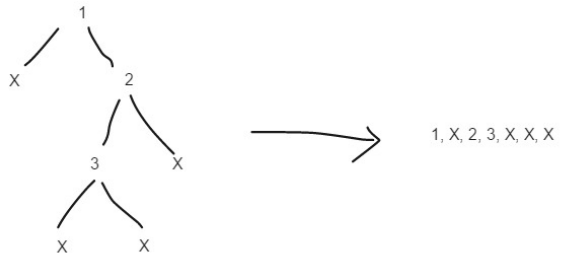
The lists does not preserve the null, so we do not have an indicator to check if a node is in the left subtree or right subtree, so 2 traversals are needed.

2. **But for this problem, we can preserve null, so we can reconstruct by just using 1 list, i.e. preorder list**

Refer to

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/discuss/74253/Easy-to-understand-Java-Solution/77362

```
public class Codec {

    private final  String spliter = ",";
    private final  String na = "X";

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {

        StringBuilder sb = new StringBuilder();
        buildString(sb, root);
        return sb.toString();
    }

    private void buildString(StringBuilder sb, TreeNode n){

        if(n == null) sb.append(na).append(spliter);
        else {
            sb.append(n.val).append(spliter);
            buildString(sb, n.left);
            buildString(sb, n.right);
        }
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {

        Queue<String> q = new LinkedList();
        q.addAll(Arrays.asList(data.split(spliter)));
        return buildTree(q);
    }

    private TreeNode buildTree(Queue<String> q){
        String val = q.poll();
        if(val.equals(na)) return null;
        else {
            TreeNode t = new TreeNode(Integer.valueOf(val));
            t.left = buildTree(q);
            t.right = buildTree(q);
            return t;
        }
    }
}
```



Solution 2: Level order traversal for Serialize using BFS (Queue) and for Deserialize using BFS (Queue) (60min)

Style 1: With "continue" statement in Serialize method

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {
    private String NA = "X";
    private String spliter = ",";

    /**
     e.g
                5                                   5
              /   \                               /     \
```

```
            3     6   => if considering NULL(x) =>  3       6
          /  \    \                                  / \     /  \
         2    4    7                                2   4    x   7
                                                   / \ / \       / \
                                                  x  x x  x      x  x
```

```
   level order serialize into string: 5,3,6,2,4,X,7,X,X,X,X,X,X,
*/
// Encodes a tree to a single string.
public String serialize(TreeNode root) {
    if(root == null) {
        return "";
    }
    StringBuilder sb = new StringBuilder();
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);
    while(!q.isEmpty()) {
        TreeNode node = q.poll();
        if(node == null) {
            sb.append(NA).append(spliter);
            // Must terminate early since node already NULL,
            // if not skip following statement then NullPointerException
            // will happen because of 'node.val' not exist
            continue;
        }
        sb.append(node.val).append(spliter);
        // Add left and right child (even if it is NULL) on queue
        q.offer(node.left);
        q.offer(node.right);
    }
    return sb.toString();
}

// Decodes your encoded data to tree.
// Decode level order traversal (5,3,6,2,4,X,7,X,X,X,X,X,X,) into tree
public TreeNode deserialize(String data) {
    if(data == "") {
        return null;
    }
    Queue<TreeNode> q = new LinkedList<TreeNode>();
```

```
        String[] values = data.split(spliter);
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        q.offer(root);
        for(int i = 1; i < values.length; i++) {
            TreeNode node = q.poll();
            if(!values[i].equals(NA)) {
                TreeNode leftNode = new TreeNode(Integer.parseInt(values[i]));
                node.left = leftNode;
                q.offer(leftNode);
            }
            i++;
            if(!values[i].equals(NA)) {
                TreeNode rightNode = new TreeNode(Integer.parseInt(values[i]));
                node.right = rightNode;
                q.offer(rightNode);
            }
        }
        return root;
    }
}
// Your Codec object will be instantiated and called as such:
// Codec ser = new Codec();
// Codec deser = new Codec();
// TreeNode ans = deser.deserialize(ser.serialize(root));

Time Complexity: O(N), where N <= 10^4 is number of nodes in the Binary Tree.
Space Complexity: O(N)
```

Style 2: Without "continue" statement in Serialize method

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
```

```
 */
public class Codec {

        private String NA = "X";
    private String spliter = ",";

    /**
     e.g
                5                                          5
              /   \                                      /     \
             3     6   => if considering NULL(x) =>    3       6
            /  \    \                                  / \     /  \
           2    4    7                                2   4   x   7
                                                     / \ / \      / \
                                                    x  x x  x    x  x

       level order serialize into string: 5,3,6,2,4,X,7,X,X,X,X,X,X,
    */
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if(root == null) {
            return "";
        }
        StringBuilder sb = new StringBuilder();
        Queue<TreeNode> q = new LinkedList<TreeNode>();
        q.offer(root);
        while(!q.isEmpty()) {
            TreeNode node = q.poll();
            if(node == null) {
                sb.append(NA).append(spliter);
                // No need continue here, simply add else branch for handling not null nodes,
                // null nodes will be auto terminate processing here
                //continue;
            } else {
                sb.append(node.val).append(spliter);
                // Add left and right child (even if it is NULL) on queue
                q.offer(node.left);
                q.offer(node.right);
            }
```

```
        }
        return sb.toString();
    }

    // Decodes your encoded data to tree.
    // Decode level order traversal (5,3,6,2,4,X,7,X,X,X,X,X,X,) into tree
    public TreeNode deserialize(String data) {
        if(data == "") {
            return null;
        }
        Queue<TreeNode> q = new LinkedList<TreeNode>();
        String[] values = data.split(spliter);
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        q.offer(root);
        for(int i = 1; i < values.length; i++) {
            TreeNode node = q.poll();
            if(!values[i].equals(NA)) {
                TreeNode leftNode = new TreeNode(Integer.parseInt(values[i]));
                node.left = leftNode;
                q.offer(leftNode);
            }
            i++;
            if(!values[i].equals(NA)) {
                TreeNode rightNode = new TreeNode(Integer.parseInt(values[i]));
                node.right = rightNode;
                q.offer(rightNode);
            }
        }
        return root;
    }
}

// Your Codec object will be instantiated and called as such:
// Codec ser = new Codec();
// Codec deser = new Codec();
// TreeNode ans = deser.deserialize(ser.serialize(root));
```

Refer to

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/discuss/74264/Short-and-straight-forward-BFS-Java-code-with-a-queueHere I use typical BFS method to handle a binary tree. I use string n to represent null values. The string of the binary tree in the example will be "1 2 3 n n 4 5 n n n n ".

When deserialize the string, I assign left and right child for each not-null node, and add the not-null children to the queue, waiting to be handled later.

```
public class Codec {
    public String serialize(TreeNode root) {
        if (root == null) return "";
        Queue<TreeNode> q = new LinkedList<>();
        StringBuilder res = new StringBuilder();
        q.add(root);
        while (!q.isEmpty()) {
            TreeNode node = q.poll();
            if (node == null) {
                res.append("n ");
                continue;
            }
            res.append(node.val + " ");
            q.add(node.left);
            q.add(node.right);
        }
        return res.toString();
    }
    public TreeNode deserialize(String data) {
        if (data == "") return null;
        Queue<TreeNode> q = new LinkedList<>();
        String[] values = data.split(" ");
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        q.add(root);
        for (int i = 1; i < values.length; i++) {
            TreeNode parent = q.poll();
            if (!values[i].equals("n")) {
                TreeNode left = new TreeNode(Integer.parseInt(values[i]));
                parent.left = left;
                q.add(left);
            }
```

```
            if (!values[++i].equals("n")) {
                TreeNode right = new TreeNode(Integer.parseInt(values[i]));
                parent.right = right;
                q.add(right);
            }
        }
        return root;
    }
}
```

Refer to

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/discuss/74264/Short-and-straight-forward-BFS-Java-code-with-a-queue/980762

Q: I also think if we are using a level order traversal, the left child is at index 2 *index + 1 and right child at index 2 * index + 2 can this info be applied in deserializing, and avoid the extra space, and do it recursively ?

A: Not able to because its difficult to update 2 root in one round for next iteration

```
    // 5,3,6,2,4,X,7,X,X,X,X,X,X,
    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if(data == "") {
            return null;
        }
        //Queue<TreeNode> q = new LinkedList<TreeNode>();
        String[] values = data.split(spliter);
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        //int index = 0;
        //q.offer(root);
        int len = values.length;
        for(int i = 0; i < values.length; i++) {
            int leftIndex = i * 2 + 1;
            int rightIndex = i * 2 + 2;
            if(leftIndex < len && !values[leftIndex].equals(NA)) {
                root.left = new TreeNode(Integer.parseInt(values[leftIndex]));
            }
```

```
            if(rightIndex < len && !values[rightIndex].equals(NA)) {
                root.right = new TreeNode(Integer.parseInt(values[rightIndex]));
            }
            // Not able to update both left / right subtree node for next iteration
            // in one for loop
        }
        return root;
    }
```

Video explain:

Serialize and Deserialize Binary Tree - Preorder Traversal - Leetcode 297 - Python

https://www.youtube.com/watch?v=u4JAi2JJhI8

Refer to

L449.Serialize and Deserialize BST (Ref.L297)