Previous Next

297. Serialize and Deserialize Binary Tree

Sept. 17, 2018 | 143.2K views

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.

Example:

```
You may serialize the following tree:

    1
   / \
  2   3
     / \
    4   5

as "[1,2,3,null,null,4,5]"
```

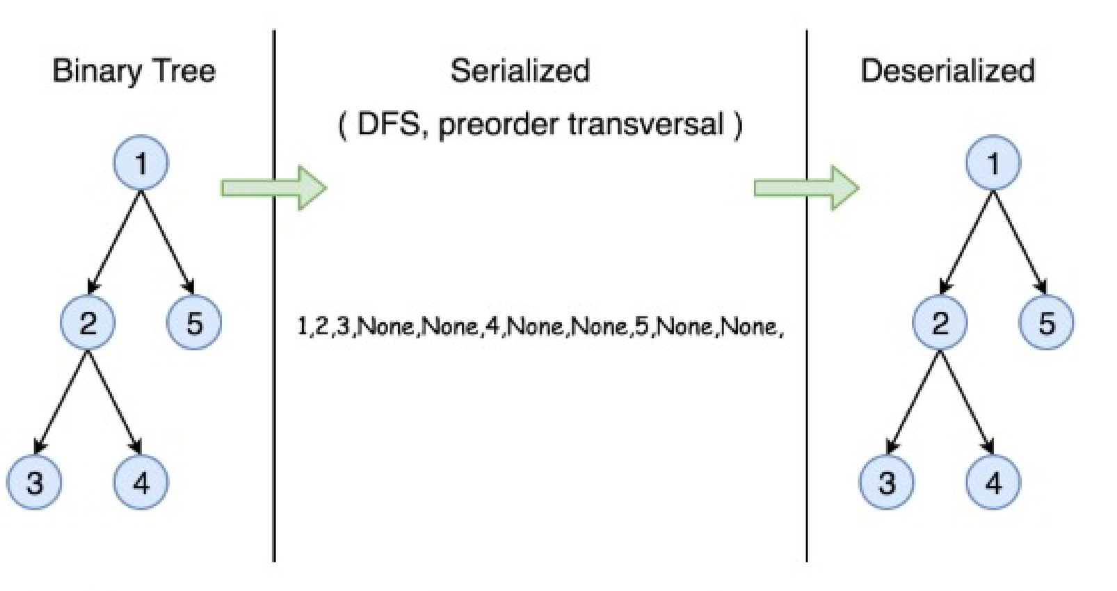
Clarification: The above format is the same as how LeetCode serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

Solution

Approach 1: Depth First Search (DFS)

Intuition



The **serialization** of a `Binary Search Tree` is essentially to encode its values and more importantly its structure. One can traverse the tree to accomplish the above task. And it is well know that we have two general strategies to do so:

- *Breadth First Search* (`BFS`)

  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

- *Depth First Search* (`DFS`)

  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

  The DFS strategy can further be distinguished as `preorder`, `inorder`, and `postorder` depending on the relative order among the root node, left node and right node.

In this task, however, the `DFS` strategy is more adapted for our needs, since the linkage among the adjacent nodes is naturally encoded in the order, which is rather helpful for the later task of **deserialization**.

Therefore, in this solution, we demonstrate an example with the `preorder` DFS strategy. One can check out more tutorial about `Binary Search Tree` on the LeetCode Explore.

Algorithm

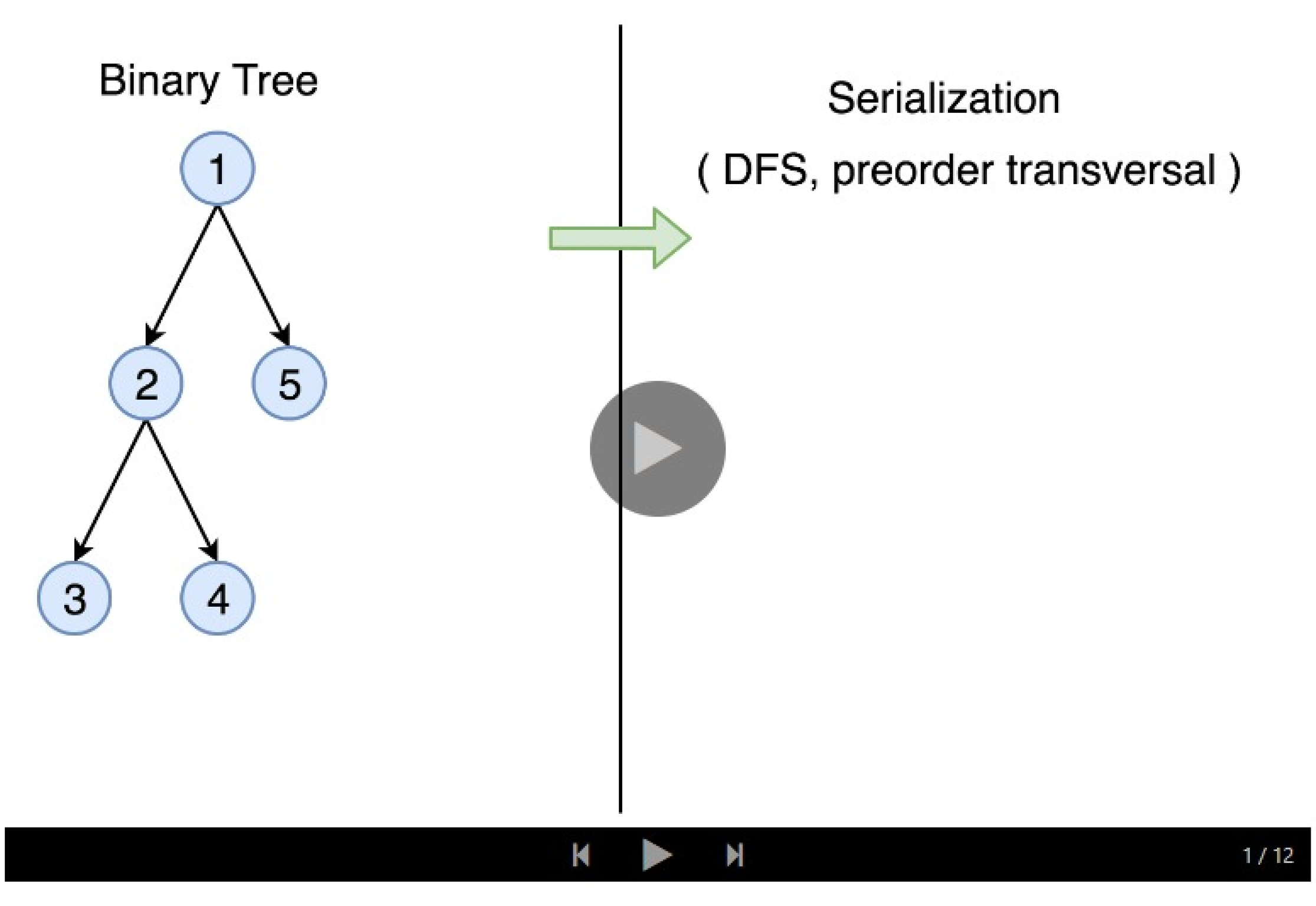
First of all, here is the definition of the `TreeNode` which we would use in the following implementation.

Java | Python | Copy

```
1  class TreeNode(object):
2      """ Definition of a binary tree node."""
3      def __init__(self, x):
4          self.val = x
5          self.left = None
6          self.right = None
```

The preorder DFS traverse follows *recursively* the order of `root -> left subtree -> right subtree`.

As an example, let's serialize the following tree. Note that serialization contains information about the node values as well as the information about the tree structure.



We start from the root, node `1`, the serialization string is `1,`. Then we jump to its left subtree with the root node `2`, and the serialization string becomes `1,2,`. Now starting from node `2`, we visit its left node `3` (`1,2,3,None,None,`) and right node `4` (`1,2,3,None,None,4,None,None`) sequentially. Note that `None,None,` appears for each leaf to mark the absence of left and right child node, this is how we save the tree structure during the serialization. And finally, we get back to the root node `1` and visit its right subtree which happens to be a leaf node `5`. Finally, the serialization string is done as `1,2,3,None,None,4,None,None,5,None,None,`.

Java | Python | Copy

```
1  # Serialization
2  class Codec:
3
4      def serialize(self, root):
5          """ Encodes a tree to a single string.
6          :type root: TreeNode
7          :rtype: str
8          """
9          def rserialize(root, string):
10             """ a recursive helper function for the serialize() function."""
11             # check base case
12             if root is None:
13                 string += 'None,'
14             else:
15                 string += str(root.val) + ','
16                 string = rserialize(root.left, string)
17                 string = rserialize(root.right, string)
18             return string
19
20         return rserialize(root, '')
```

Now let's deserialize the serialization string constructed above `1,2,3,None,None,4,None,None,5,None,None,`. It goes along the string, initiate the node value and then calls itself to construct its left and right child nodes.

Java | Python | Copy

```
1  # Deserialization
2  class Codec:
3
4      def deserialize(self, data):
5          """Decodes your encoded data to tree.
6          :type data: str
7          :rtype: TreeNode
8          """
9          def rdeserialize(l):
10             """ a recursive helper function for deserialization."""
11             if l[0] == 'None':
12                 l.pop(0)
13                 return None
14
15             root = TreeNode(l[0])
16             l.pop(0)
17             root.left = rdeserialize(l)
18             root.right = rdeserialize(l)
19             return root
20
21         data_list = data.split(',')
22         root = rdeserialize(data_list)
23         return root
```

Complexity Analysis

- Time complexity : in both serialization and deserialization functions, we visit each node exactly once, thus the time complexity is $O(N)$, where N is the number of nodes, *i.e.* the size of tree.

- Space complexity : in both serialization and deserialization functions, we keep the entire tree, either at the beginning or at the end, therefore, the space complexity is $O(N)$.

The solutions with BFS or other DFS strategies normally will have the same time and space complexity.

Further Space Optimization

In the above solution, we store the node value and the references to `None` child nodes, which means $N \cdot V + 2N$ complexity, where V is the size of value. That is called *natural serialization*, and has was implemented above.

The $N \cdot V$ component here is the encoding of values, can't be optimized further, but there is a way to reduce $2N$ part which is the encoding of the tree structure.

The number of unique binary tree structures that can be constructed using `n` nodes is $C(n)$, where $C(n)$ is the `nth` Catalan number. Please refer to this article for more information.

There are $C(n)$ possible structural configurations of a binary tree with n nodes, so the largest index value that we might need to store is $C(n) - 1$. That means storing the index value could require up to 1 bit for $n \leq 2$, or $\lceil log_2(C(n) - 1) \rceil$ bits for $n > 2$.

In this way one could reduce the encoding of the tree structure by $\log N$. More precisely, the Catalan numbers grow as $C(n) \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$ and hence the theoretical minimum of storage for the tree structure that could be achieved is $log(C(n)) \sim 2n - \frac{3}{2}\log(n) - \frac{1}{2}\log(\pi)$

Rate this article: ★★★★★

Previous Next

Comments: 52

Sort By

Type comment here... (Markdown is supported)

Preview Post

zjlvmiao ★ 307 December 20, 2018 11:07 AM

BFS with Queue

```
class Codec:
    '''      O(n) time and O(n) space, BFS traversal
```

Read More

76 Share Reply

SHOW 6 REPLIES

ZitaoWang ★ 1346 January 1, 2019 7:04 AM

I think the `serialize` method given in the solution is not `O(N)`, because in the line `string += str(root.val) + ','`, one needs to create a copy of `string` first because they are immutable. Hence the worst case runtime is `O(N^2)`. With a small tweak of the original idea, one can achieve `O(N)` runtime, as follows:

Read More

37 Share Reply

SHOW 5 REPLIES

dsc5085 ★ 52 October 4, 2018 6:58 AM

i wish they explained the BFS way since thats how leetcode does it

22 Share Reply

SHOW 2 REPLIES

SanD91 ★ 542 February 9, 2019 12:06 PM

Leetcode's BFS serialization. Easier to understand.

```
public class Solution {
// Encodes a tree to a single string.
```

Read More

9 Share Reply

SHOW 3 REPLIES

Inmlv ★ 68 September 27, 2018 8:07 PM

Two questions:

1. The problem asks for "converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer". While this solution could pass leetcode's tests, is it what the problem is asking? You're serializing the string representation of the integer, which is not a

Read More

8 Share Reply

SHOW 4 REPLIES

inctrl ★ 77 November 17, 2018 1:07 PM

What if the input is like this?
1
/
null 2

Read More

4 Share Reply

SHOW 2 REPLIES

kuhi ★ 11 September 20, 2018 7:37 AM

I believe removing 0th Index element from the list in deserialization in line 9, i.e- `l.remove(0);`
should also be for the condition line 4 for it to work: `l.get(0).equals("null")`

4 Share Reply

SHOW 1 REPLY

ZitaoWang ★ 1346 January 2, 2019 6:20 AM

A solution with iterative preorder traversal for `deserialize`:

```
class Codec:
```

Read More

5 Share Reply

SHOW 1 REPLY

aveekbiswas ★ 17 October 22, 2018 8:57 PM

The string constructed in the serialize function is "1,2,null,null,3,4,null,null,5,null,null," and not "1,2,3,null,null,4,null,null,5,null,null," as mentioned above.

3 Share Reply

SHOW 2 REPLIES

qinlei515 ★ 230 November 24, 2018 9:31 AM

NV+2N, how comes the 2N part?

2 Share Reply

SHOW 2 REPLIES

‹ 1 2 3 4 5 6 ›