

Problem 297: Serialize and Deserialize Binary Tree

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

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.

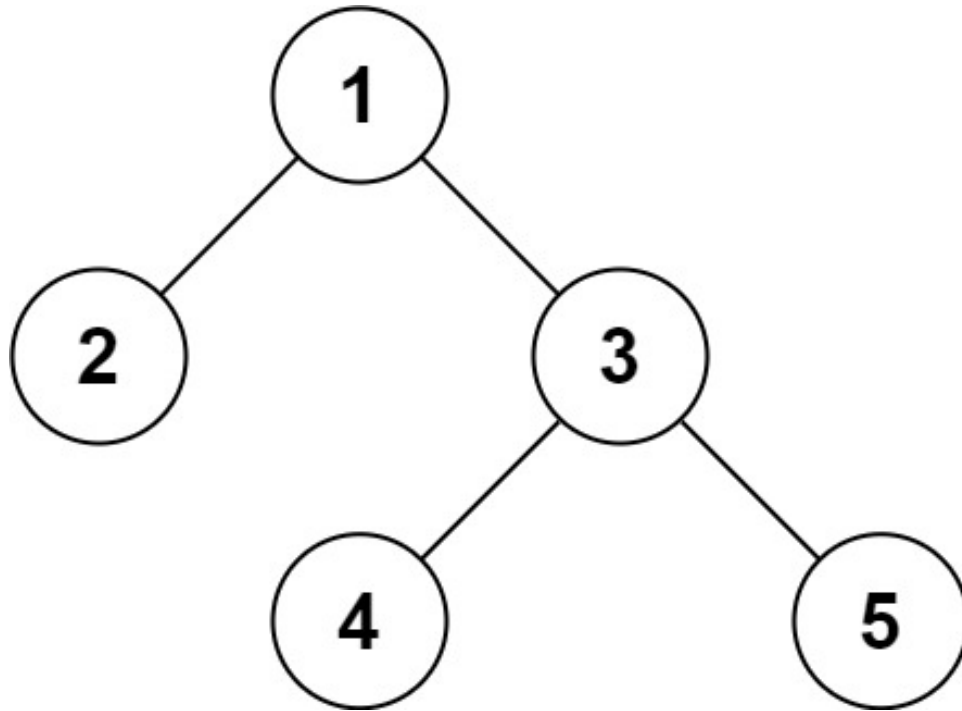
Clarification:

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

Code Snippets

C++:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:

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

    }

    // Decodes your encoded data to tree.
    TreeNode* deserialize(string data) {

    }
};

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

Java:

```

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

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

    }

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

    }

}

// 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));

```

Python3:

```

# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str

```

```

"""

def deserialize(self, data):
    """Decodes your encoded data to tree.

    :type data: str
    :rtype: TreeNode
    """

# Your Codec object will be instantiated and called as such:
# ser = Codec()
# deser = Codec()
# ans = deser.deserialize(ser.serialize(root))

```

Python:

```

# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """

    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """

```

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

JavaScript:

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *   this.val = val;
 *   this.left = this.right = null;
 * }
 */

/**
 * Encodes a tree to a single string.
 *
 * @param {TreeNode} root
 * @return {string}
 */
var serialize = function(root) {

};

/**
 * Decodes your encoded data to tree.
 *
 * @param {string} data
 * @return {TreeNode}
 */
var deserialize = function(data) {

};

/**
 * Your functions will be called as such:
 * deserialize(serialize(root));
 */
```

TypeScript:

```
/**
 * Definition for a binary tree node.
 * class TreeNode {
 *   val: number
 *   left: TreeNode | null
 *   right: TreeNode | null
 *   constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 *   {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 *   }
 * }
 */

/**
 * Encodes a tree to a single string.
 */
function serialize(root: TreeNode | null): string {

};

/**
 * Decodes your encoded data to tree.
 */
function deserialize(data: string): TreeNode | null {

};

/**
 * Your functions will be called as such:
 * deserialize(serialize(root));
 */
```

C#:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *   public int val;
```

```

* public TreeNode left;
* public TreeNode right;
* public TreeNode(int x) { val = x; }
* }
*/
public class Codec {

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

    }

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

    }
}

// 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));

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
/** Encodes a tree to a single string. */
char* serialize(struct TreeNode* root) {

}

/** Decodes your encoded data to tree. */
struct TreeNode* deserialize(char* data) {

}

```



```
// Your functions will be called as such:  
// char* data = serialize(root);  
// deserialize(data);
```

Go:

```
/**  
 * Definition for a binary tree node.  
 * type TreeNode struct {  
 *     Val int  
 *     Left *TreeNode  
 *     Right *TreeNode  
 * }  
 */  
  
type Codec struct {  
  
}  
  
func Constructor() Codec {  
  
}  
  
// Serializes a tree to a single string.  
func (this *Codec) serialize(root *TreeNode) string {  
  
}  
  
// Deserializes your encoded data to tree.  
func (this *Codec) deserialize(data string) *TreeNode {  
  
}  
  
/**  
 * Your Codec object will be instantiated and called as such:  
 * ser := Constructor();  
 * deser := Constructor();  
 * data := ser.serialize(root);  
 * ans := deser.deserialize(data);  
 */
```

Kotlin:

```
/**
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */

class Codec() {
    // Encodes a URL to a shortened URL.
    fun serialize(root: TreeNode?): String {

    }

    // Decodes your encoded data to tree.
    fun deserialize(data: String): TreeNode? {

    }
}

/**
 * Your Codec object will be instantiated and called as such:
 * var ser = Codec()
 * var deser = Codec()
 * var data = ser.serialize(longUrl)
 * var ans = deser.deserialize(data)
 */
```

Swift:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public var val: Int
 *     public var left: TreeNode?
 *     public var right: TreeNode?
 *     public init(_ val: Int) {
 *         self.val = val
 *         self.left = nil
 *         self.right = nil
 *     }
 * }
```

```

* }
* }
*/

class Codec {
func serialize(_ root: TreeNode?) -> String {

}

func deserialize(_ data: String) -> TreeNode? {

}
}

// Your Codec object will be instantiated and called as such:
// var ser = Codec()
// var deser = Codec()
// deser.deserialize(ser.serialize(root))

```

Rust:

```

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//     pub val: i32,
//     pub left: Option<Rc<RefCell<TreeNode>>>,
//     pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
//     #[inline]
//     pub fn new(val: i32) -> Self {
//         TreeNode {
//             val,
//             left: None,
//             right: None
//         }
//     }
// }

use std::rc::Rc;
use std::cell::RefCell;

struct Codec {

```

```

}

/**
 * `&self` means the method takes an immutable reference.
 * If you need a mutable reference, change it to `&mut self` instead.
 */
impl Codec {
  fn new() -> Self {

  }

  fn serialize(&self, root: Option<Rc<RefCell<TreeNode>>>) -> String {

  }

  fn deserialize(&self, data: String) -> Option<Rc<RefCell<TreeNode>>> {

  }
}

/**
 * Your Codec object will be instantiated and called as such:
 * let obj = Codec::new();
 * let data: String = obj.serialize(strs);
 * let ans: Option<Rc<RefCell<TreeNode>>> = obj.deserialize(data);
 */

```

Ruby:

```

# Definition for a binary tree node.
# class TreeNode
#   attr_accessor :val, :left, :right
#   def initialize(val)
#     @val = val
#     @left, @right = nil, nil
#   end
# end

# Encodes a tree to a single string.
#
# @param {TreeNode} root

```

```

# @return {string}
def serialize(root)

end

# Decodes your encoded data to tree.
#
# @param {string} data
# @return {TreeNode}
def deserialize(data)

end

# Your functions will be called as such:
# deserialize(serialize(data))

```

PHP:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($value) { $this->val = $value; }
 * }
 */

class Codec {
function __construct() {

}

/**
 * @param TreeNode $root
 * @return String
 */
function serialize($root) {

}
}

```

```

/**
 * @param String $data
 * @return TreeNode
 */
function deserialize($data) {

}

}

/**
 * Your Codec object will be instantiated and called as such:
 * $ser = Codec();
 * $deser = Codec();
 * $data = $ser->serialize($root);
 * $ans = $deser->deserialize($data);
 */

```

Scala:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(var _value: Int) {
 *   var value: Int = _value
 *   var left: TreeNode = null
 *   var right: TreeNode = null
 * }
 */

class Codec {
  // Encodes a list of strings to a single string.
  def serialize(root: TreeNode): String = {

  }

  // Decodes a single string to a list of strings.
  def deserialize(data: String): TreeNode = {

  }
}

/**
 * Your Codec object will be instantiated and called as such:

```

```

* var ser = new Codec()
* var deser = new Codec()
* val s = ser.serialize(root)
* val ans = deser.deserialize(s)
*/

```

Solutions

C++ Solution:

```

/*
* Problem: Serialize and Deserialize Binary Tree
* Difficulty: Hard
* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
*   int val;
*   TreeNode *left;
*   TreeNode *right;
*   TreeNode(int x) : val(x), left(NULL), right(NULL) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
class Codec {
public:

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

}

```

```

// Decodes your encoded data to tree.
TreeNode* deserialize(string data) {

}

};

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

```

Java Solution:

```

/**
 * Problem: Serialize and Deserialize Binary Tree
 * Difficulty: Hard
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

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

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

    }

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

```



```

}
}

// 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));

```

Python3 Solution:

```

# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """

    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """

# Your Codec object will be instantiated and called as such:
# ser = Codec()
# deser = Codec()
# ans = deser.deserialize(ser.serialize(root))

```

Python Solution:

```

# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """

    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """

# Your Codec object will be instantiated and called as such:
# ser = Codec()
# deser = Codec()
# ans = deser.deserialize(ser.serialize(root))

```

JavaScript Solution:

```

/**
 * Problem: Serialize and Deserialize Binary Tree
 * Difficulty: Hard
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

```

```

/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *   this.val = val;
 *   this.left = this.right = null;
 * }
 */

/**
 * Encodes a tree to a single string.
 *
 * @param {TreeNode} root
 * @return {string}
 */
var serialize = function(root) {

};

/**
 * Decodes your encoded data to tree.
 *
 * @param {string} data
 * @return {TreeNode}
 */
var deserialize = function(data) {

};

/**
 * Your functions will be called as such:
 * deserialize(serialize(root));
 */

```

TypeScript Solution:

```

/**
 * Problem: Serialize and Deserialize Binary Tree
 * Difficulty: Hard
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 */

```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* class TreeNode {
*   val: number
*   left: TreeNode | null
*   right: TreeNode | null
*   constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
*   {
*     this.val = (val===undefined ? 0 : val)
*     this.left = (left===undefined ? null : left)
*     this.right = (right===undefined ? null : right)
*   }
* }
*/

/*
* Encodes a tree to a single string.
*/
function serialize(root: TreeNode | null): string {

};

/*
* Decodes your encoded data to tree.
*/
function deserialize(data: string): TreeNode | null {

};

/**
* Your functions will be called as such:
* deserialize(serialize(root));
*/

```

C# Solution:

```

/*
 * Problem: Serialize and Deserialize Binary Tree
 * Difficulty: Hard
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

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

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

    }

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

    }

}

// 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));

```

C Solution:

```

/*
 * Problem: Serialize and Deserialize Binary Tree
 * Difficulty: Hard

```

```

* Tags: string, tree, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
*   int val;
*   struct TreeNode *left;
*   struct TreeNode *right;
* };
*/
/** Encodes a tree to a single string. */
char* serialize(struct TreeNode* root) {

}

/** Decodes your encoded data to tree. */
struct TreeNode* deserialize(char* data) {

}

// Your functions will be called as such:
// char* data = serialize(root);
// deserialize(data);

```

Go Solution:

```

// Problem: Serialize and Deserialize Binary Tree
// Difficulty: Hard
// Tags: string, tree, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

/**
* Definition for a binary tree node.

```

```

* type TreeNode struct {
*   Val int
*   Left *TreeNode
*   Right *TreeNode
* }
*/

type Codec struct {

}

func Constructor() Codec {

}

// Serializes a tree to a single string.
func (this *Codec) serialize(root *TreeNode) string {

}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {

}

/**
 * Your Codec object will be instantiated and called as such:
 * ser := Constructor();
 * deser := Constructor();
 * data := ser.serialize(root);
 * ans := deser.deserialize(data);
 */

```

Kotlin Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *   var left: TreeNode? = null
 *   var right: TreeNode? = null

```

```

* }
*/

class Codec() {
    // Encodes a URL to a shortened URL.
    fun serialize(root: TreeNode?): String {

    }

    // Decodes your encoded data to tree.
    fun deserialize(data: String): TreeNode? {

    }
}

/**
 * Your Codec object will be instantiated and called as such:
 * var ser = Codec()
 * var deser = Codec()
 * var data = ser.serialize(longUrl)
 * var ans = deser.deserialize(data)
 */

```

Swift Solution:

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public var val: Int
 *     public var left: TreeNode?
 *     public var right: TreeNode?
 *     public init(_ val: Int) {
 *         self.val = val
 *         self.left = nil
 *         self.right = nil
 *     }
 * }
 */

class Codec {
    func serialize(_ root: TreeNode?) -> String {

```



```

}

func deserialize(_ data: String) -> TreeNode? {

}

}

// Your Codec object will be instantiated and called as such:
// var ser = Codec()
// var deser = Codec()
// deser.deserialize(ser.serialize(root))

```

Rust Solution:

```

// Problem: Serialize and Deserialize Binary Tree
// Difficulty: Hard
// Tags: string, tree, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//     pub val: i32,
//     pub left: Option<Rc<RefCell<TreeNode>>>,
//     pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
//     #[inline]
//     pub fn new(val: i32) -> Self {
//         TreeNode {
//             val,
//             left: None,
//             right: None
//         }
//     }
// }

```

```

use std::rc::Rc;
use std::cell::RefCell;
struct Codec {

}

/**
 * `&self` means the method takes an immutable reference.
 * If you need a mutable reference, change it to `&mut self` instead.
 */
impl Codec {
fn new() -> Self {

}

fn serialize(&self, root: Option<Rc<RefCell<TreeNode>>>) -> String {

}

fn deserialize(&self, data: String) -> Option<Rc<RefCell<TreeNode>>> {

}

}

/**
 * Your Codec object will be instantiated and called as such:
 * let obj = Codec::new();
 * let data: String = obj.serialize(strs);
 * let ans: Option<Rc<RefCell<TreeNode>>> = obj.deserialize(data);
 */

```

Ruby Solution:

```

# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val)
# @val = val
# @left, @right = nil, nil
# end
# end

```

```

# Encodes a tree to a single string.
#
# @param {TreeNode} root
# @return {string}
def serialize(root)

end

# Decodes your encoded data to tree.
#
# @param {string} data
# @return {TreeNode}
def deserialize(data)

end

# Your functions will be called as such:
# deserialize(serialize(data))

```

PHP Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($value) { $this->val = $value; }
 * }
 */

class Codec {
function __construct() {

}

/**
 * @param TreeNode $root
 * @return String

```

```

*/
function serialize($root) {

}

/**
 * @param String $data
 * @return TreeNode
 */
function deserialize($data) {

}

}

/**
 * Your Codec object will be instantiated and called as such:
 * $ser = Codec();
 * $deser = Codec();
 * $data = $ser->serialize($root);
 * $ans = $deser->deserialize($data);
 */

```

Scala Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(var _value: Int) {
 *   var value: Int = _value
 *   var left: TreeNode = null
 *   var right: TreeNode = null
 * }
 */

class Codec {
  // Encodes a list of strings to a single string.
  def serialize(root: TreeNode): String = {

  }

  // Decodes a single string to a list of strings.
  def deserialize(data: String): TreeNode = {

```

```
}  
}
```

```
/**
```

```
 * Your Codec object will be instantiated and called as such:
```

```
 * var ser = new Codec()
```

```
 * var deser = new Codec()
```

```
 * val s = ser.serialize(root)
```

```
 * val ans = deser.deserialize(s)
```

```
 */
```