Rust 程序语言设计 Assignment 3

姓名：向胤兴

班级：计算机2102

学号：2215012469

题目 1：二叉树的层序遍历

题目描述：

请你实现二叉树结构，并从根节点开始，返回其节点值的层序遍历（逐层地，从左到右返回所有节点）。

测试用例 1：

输入：`root = [1]`

输出：`[[1]]`

测试用例 2：

输入：`root = [3,9,20,null,null,15,7]`

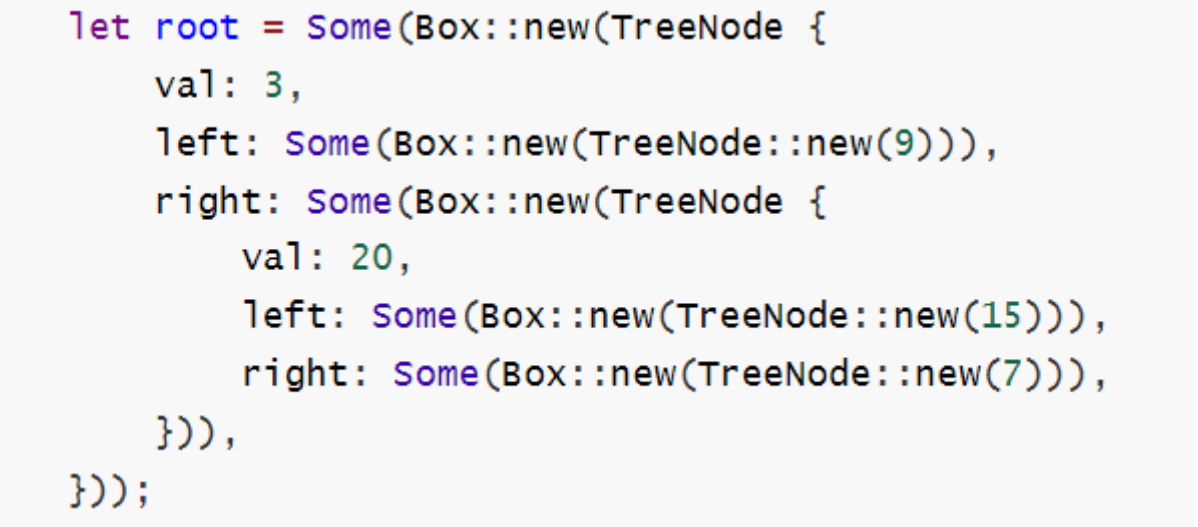
输出：`[[3],[9,20],[15,7]]`

代码实现：

```rust
use std::collections::VecDeque;

#[derive(Debug, PartialEq, Eq)]
pub struct TreeNode {
    pub val: i32,
    pub left: Option<Box<TreeNode>>,
    pub right: Option<Box<TreeNode>>,
}
//定义树节点
impl TreeNode {
    #[inline]
    pub fn new(val: i32) -> Self {
        TreeNode { val, left: None, right: None }
    }
}
//按序处理
pub fn level_order(root: Option<Box<TreeNode>>) -> Vec<Vec<i32>> {
    let mut res = Vec::new();
    if root.is_none() {
        return res;
    }

    let mut queue = VecDeque::new();
```

```
    queue.push_back(root);

    while !queue.is_empty() {
        let level_size = queue.len();
        let mut level = Vec::new();
        for _ in 0..level_size {
            if let Some(Some(node)) = queue.pop_front() {
                level.push(node.val);
                if node.left.is_some() {
                    queue.push_back(node.left);
                }
                if node.right.is_some() {
                    queue.push_back(node.right);
                }
            }
        }
        res.push(level);
    }
    res
}

fn main() {
    let root = Some(Box::new(TreeNode {
        val: 3,
        left: Some(Box::new(TreeNode::new(9))),
        right: Some(Box::new(TreeNode {
            val: 20,
            left: Some(Box::new(TreeNode::new(15))),
            right: Some(Box::new(TreeNode::new(7))),
        })),
    }));

    let result = level_order(root);
    println!("{:?}", result);
}
```

控制台输出截图：



```
   Compiling project1 v0.1.0 (D:\code\rust\project1)
    Finished dev [unoptimized + debuginfo] target(s) in 0.54s
     Running `target\debug\project1.exe`
[[3], [9, 20], [15, 7]]
```

代码分析：

该算法利用广度优先搜索（BFS）来实现二叉树的层序遍历。通过队列 `VecDeque` 来存储每一层的节点，并逐层访问每一个节点。每次从队列中取出一个节点，将其值加入当前层的结果中，如果该节点有子节点，则将其子节点加入队列中。最后，将每一层的结果按顺序存入返回结果中。

题目 2：计算岛屿的数量

题目描述：

给你一个由 '1'（陆地）和 '0'（水）组成的二维网格，请你计算网格中岛屿的数量。岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。此外，你可以假设该网格的四条边均被水包围。

测试用例 1：

输入：

```
grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
```

输出：`1`

测试用例 2：

输入：

```
grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
```

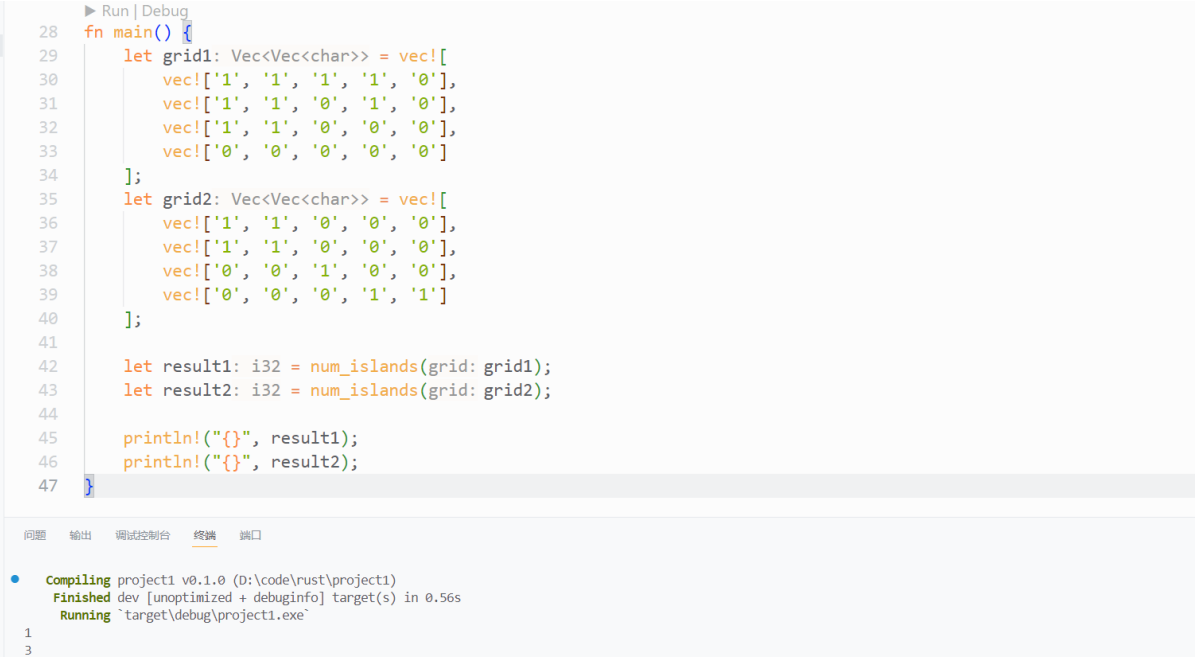
输出：`3`

代码实现：

```rust
pub fn num_islands(grid: Vec<Vec<char>>) -> i32 {
    let mut grid = grid.clone();
    let mut count = 0;
    //深度优先，把临接的陆地全部消除
```

```rust
    fn dfs(grid: &mut Vec<Vec<char>>, i: usize, j: usize) {
        if i >= grid.len() || j >= grid[0].len() || grid[i][j] == '0' {
            return;
        }
        grid[i][j] = '0';
        if i > 0 { dfs(grid, i - 1, j); }
        if j > 0 { dfs(grid, i, j - 1); }
        dfs(grid, i + 1, j);
        dfs(grid, i, j + 1);
    }

    for i in 0..grid.len() {
        for j in 0..grid[0].len() {
            if grid[i][j] == '1' {
                count += 1;
                dfs(&mut grid, i, j);
            }
        }
    }

    count
}

fn main() {
    let grid1 = vec![
        vec!['1', '1', '1', '1', '0'],
        vec!['1', '1', '0', '1', '0'],
        vec!['1', '1', '0', '0', '0'],
        vec!['0', '0', '0', '0', '0']
    ];
    let grid2 = vec![
        vec!['1', '1', '0', '0', '0'],
        vec!['1', '1', '0', '0', '0'],
        vec!['0', '0', '1', '0', '0'],
        vec!['0', '0', '0', '1', '1']
    ];

    let result1 = num_islands(grid1);
    let result2 = num_islands(grid2);

    println!("{}", result1);
    println!("{}", result2);
}
```

控制台输出截图：



```
▶ Run | Debug
fn main() {
    let grid1: Vec<Vec<char>> = vec![
        vec!['1', '1', '1', '1', '0'],
        vec!['1', '1', '0', '1', '0'],
        vec!['1', '1', '0', '0', '0'],
        vec!['0', '0', '0', '0', '0']
    ];
    let grid2: Vec<Vec<char>> = vec![
        vec!['1', '1', '0', '0', '0'],
        vec!['1', '1', '0', '0', '0'],
        vec!['0', '0', '1', '0', '0'],
        vec!['0', '0', '0', '1', '1']
    ];

    let result1: i32 = num_islands(grid: grid1);
    let result2: i32 = num_islands(grid: grid2);

    println!("{}", result1);
    println!("{}", result2);
}
```

```
问题  输出  调试控制台  终端  端口

  Compiling project1 v0.1.0 (D:\code\rust\project1)
   Finished dev [unoptimized + debuginfo] target(s) in 0.56s
    Running `target\debug\project1.exe`
1
3
```

代码分析：

该算法使用深度优先搜索（DFS）来计算岛屿的数量。对于每一个网格中的 `1`，即陆地，将其转换为 `0` 并使用 DFS 遍历所有相邻的陆地。每找到一个新的 `1`，即表示发现一个新的岛屿，岛屿计数器加一。通过对每个网格进行检查，最终得到岛屿的总数。

题目 3：解码字符串

题目描述：

给定一个经过编码的字符串，返回它解码后的字符串。编码规则为 `k[encoded_string]`，表示其中方括号内部的 `encoded_string` 正好重复 `k` 次。注意 `k` 保证为正整数。你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。此外，你可以认为原始数据不含数字，所有的数字只表示重复的次数 `k`，例如不会出现像 `3a` 或 `2[4]` 的输入。

测试用例 1：

输入：`s = "3[a]2[bc]"`

输出：`"aaabcbc"`

测试用例 2：

输入：`s = "3[a2[c]]"`

输出：`"accaccacc"`

测试用例 3：

输入：`s = "2[abc]3[cd]ef"`

输出：`"abcabccdcdcdef"`

测试用例 4：

输入：`s = "abc3[cd]xyz"`

输出：`"abccdcdcdxyz"`

代码实现：

```rust
pub fn decode_string(s: String) -> String {
    let mut stack: Vec<(String, usize)> = Vec::new();
    let mut current_str = String::new();
    let mut k = 0;
    //依次判断处理
    for c in s.chars() {
        if c.is_digit(10) {
            k = k * 10 + c.to_digit(10).unwrap() as usize;
        } else if c == '[' {
            stack.push((current_str, k));
            current_str = String::new();
            k = 0;
        } else if c == ']' {
            if let Some((prev_str, count)) = stack.pop() {
                current_str = prev_str + &current_str.repeat(count);
            }
        } else {
            current_str.push(c);
        }
    }

    current_str
}

fn main() {
    let s1 = "3[a]2[bc]".to_string();
    let s2 = "3[a2[c]]".to_string();
    let

 s3 = "2[abc]3[cd]ef".to_string();
    let s4 = "abc3[cd]xyz".to_string();

    println!("{}", decode_string(s1));
    println!("{}", decode_string(s2));
    println!("{}", decode_string(s3));
    println!("{}", decode_string(s4));
}
```

控制台输出截图：

```
▶ Run | Debug
fn main() {
    let s1: String = "3[a]2[bc]".to_string();
    let s2: String = "3[a2[c]]".to_string();
    let s3: String = "2[abc]3[cd]ef".to_string();
    let s4: String = "abc3[cd]xyz".to_string();

    println!("{}", decode_string(s1));
    println!("{}", decode_string(s2));
    println!("{}", decode_string(s3));
    println!("{}", decode_string(s4));
}
```

```
问题  输出  调试控制台  终端  端口

   Compiling project1 v0.1.0 (D:\code\rust\project1)
    Finished dev [unoptimized + debuginfo] target(s) in 0.43s
     Running `target\debug\project1.exe`
aaabcbc
accaccacc
abcabccdcdcdef
abccdcdcdxyz
```

代码分析：

该算法使用栈来处理嵌套的编码字符串。遍历字符串，当遇到数字时，计算出重复的次数 `k`；遇到 `[` 时，将当前的字符串和 `k` 入栈；遇到 `]` 时，从栈中弹出前一个字符串和 `k`，将当前字符串重复 `k` 次并附加到前一个字符串上；其他字符则直接附加到当前字符串中。最终得到解码后的字符串。