HANDS ON 1 REPORT

Competitive Programming and Contests

SALMI ABDERRAHMANE

a.salmi1@studenti.unipi.it

Exercise 1: Checking if a Binary Tree is a Binary Search Tree

A binary tree is a BST when the following conditions are met for each node:

- The value of the left child is less than the node's value
- The value of the right child is greater than the node's value

Approach

To solve this problem, we used a recursive function.

The main function “is_bst” calls the recursive function “is_bst_rec” and gives it the root node as a parameter, and sets the min and max constraints as None as an initial value.

```rust
pub fn is_bst(&self) -> bool {
        self.is_bst_rec(Some(0), None, None)
    }
```

The recursive function “is_bst_rec” checks if the current node satisfies the BST conditions by comparing its value to the min and max constraints.

If these conditions are not met, the fun will return “false”, which indicates that the tree is not a BST tree.

```rust
// Check if the current node satisfies the BST properties
if let Some(min_val) = min {
    if node.key <= min_val {
        return false;
    }
}
if let Some(max_val) = max {
    if node.key >= max_val {
        return false;
    }
}
```

If the conditions are met for the current node, the fun will recursively check if they are also met for the left and right subtrees by re-calling itself and updating the parameters, which are the node id, the min, and the max.

```
// Check if the left and right subtrees are BST
let is_left_bst = self.is_bst_rec(node.id_left, min, Some(node.key));
let is_right_bst = self.is_bst_rec(node.id_right, Some(node.key), max);

is_left_bst && is_right_bst
```

The recursive function will stop when we meet a node with the value “None”, which indicates that the tree is empty.

Exercise 2: Maximum Path Sum Connecting Two Leaves

The goal is to find the maximum path sum that connects two leaf nodes in a binary tree.

Approach

Like the previous problem, we used a recursive function to solve this problem as well.

The main function “max_path_sum” calls the recursive function “max_path_sum_rec” and gives it the root node as a parameter, and sets the current max path sum value to MIN as an initial value.

```
pub fn max_path_sum(&self) -> u32 {
  let mut max_sum = u32::MIN;
  self.max_path_sum_rec(Some(0), &mut max_sum);
  max_sum
}
```

The recursive function “max_path_sum_rec” checks if the current node is a leaf (meaning it has no left or right children).

If it’s the case, it’ll return the node’s key as the base case.

Otherwise, it’ll recursively calculate the max path sum for both the left and right subtrees (if they exist).

```
let left_sum = if let Some(left_id) = node.id_left {
    self.max_path_sum_rec(Some(left_id), max_sum)
} else {
    0

// Calculate the sum of the right subtree (if it exists)
let right_sum = if let Some(right_id) = node.id_right {
    self.max_path_sum_rec(Some(right_id), max_sum)
} else {
    0
};
```

If the node has both left and right children, in that case we’ll calculate the current max path sum by summing the current node’s key plus the sums of its left and right subtrees.

Then the current max path sum is compared to the best max sum we found so far (max_sum), if it’s greater than it then we update its value.

```
if node.id_left.is_some() && node.id_right.is_some() {
    let current_sum = left_sum + node.key + right_sum;
    if current_sum > *max_sum {
        *max_sum = current_sum;
    }
}
```

Finally the fun returns the maximum path sum from the current node to one of its leaves (either from the left or right subtree) for its parent to use.