1 Routings in transformation program

```
input  : program, program to be processed; ast, abstact syntax tree of the program
output: new_program, processed program

new_program ← ∅
foreach block ∈ program do
    new_block ← ∅
    pointers[ ] ← ∅
    eval_prog ← Construct a random evaluation program from the evaluation templates
    block ← Mix_Program(block, eval_prog)
    constant_pool ← [All constants in the block]
    CT ← build_constant_trees(constant_pool, pointers[ ])
    CT_val ← decode(CT)
    append(new_block, dynamic_build_constant_tree(CT_val))
    foreach statement ∈ block do
        if have_constant(statement, ast) then
            foreach constant ∈ statement do
                pointer ← locate_subtree(pointers[ ], constant)
                substutite(statement, constant, decode(pointer))
            end
        end
        val ← left value of the statement
        append(new_block, statement)
        append(new_block, update_graph(CT, val))
    end
    append(new_program, new_block)
end
return new_program
```

Algorithm 1: Overview

```
input  : constant, constant to be search in the tree
         pointers[ ], a set of pointers used to point to the
         root of the subgraphs representing the constant
output: pointer, the root of the subgraph representing the constant
         NULL, if not find

if constant ∈ pointers[ ] then
    return pointer
else
    return NULL
end
```

Algorithm 2: Locate subtree

```
input  : constant_pool, a set of constants used in the block;
         pointers[ ], a set of pointers used to point to the
         root of the subgraphs representing the constant
output: CT, the root of the constant tree
CT ← Root of init_tree
foreach constant ∈ constant_pool do
    subtree_root ← build_const_tree(constant)
    node ← locate_subtree(pointers[ ], constant)
    if node == NULL then //if cannot find one, construct one
        node ← random_node(CT)
        merge_tree(node, subtree_root)
    else//if find one, make a duplication in probability
        toss a coin
        if head is up then
            node ← random_node(CT)
            merge_tree(node, subtree_root)
        end
    end
    pointers[ ].join((node, constant ))
end
return CT
```

Algorithm 3: Build Constant Trees

2 Routings in watermarked program

```
input  : CT, the root of the constant tree; val, value update refers to
output: None
node ← find_upgradable_node(CT)
if node == node →parent →left then
    parent_point ← &node →parent →left
else
    parent_point ← &node →parent →right
end
if val is odd then //left rotate
    switch rotate case do
        case 1 left_rotate_1()
        case 2 left_rotate_2()
        case 3 left_rotate_3()
        case 4 left_rotate_4()
    end
else
    switch rotate case do
        case 1 right_rotate_1()
        case 2 right_rotate_2()
        case 3 right_rotate_3()
        case 4 right_rotate_4()
    end
end
if CT →left ≠ CT →right then //update the child pointers of the root
    CT →right ← CT →left
end
```

Algorithm 4: UpdateGraph

```
*parent_point ← node →right;
node →right →parent ← node →parent;
node →parent ← node →left;
node →right ← node →parent →left;
node →parent →left →parent ← node;
node →parent →left ← node;
```

Algorithm 5: Left rotate case 1

```
allocate(new_node);
new_node →left ← node →right →left;
new_node →right ← node →right →right;
node →right →left →parent ← new_node;
node →right →right →parent ← new_node;
node →right →right ← new_node;
node →right →left ← node;
node →right →parent ← node →parent;
*parent_point ← node →right;
node →parent ← node →right;
allocate(another_new_node);
node →right ← another_new_node;
pointers[ ].update(node →parent, node →parent →right);
```

Algorithm 6: Left rotate case 2

```
∗parent_point ← node →left;
node →left →parent ← node →parent;
node →parent ← node →right;
node →left ← node →parent →right;
node →parent →right →parent ← node;
node →parent →right ← node;
```

Algorithm 7: Right rotate case 1

```
allocate(new_node);
new_node →left ← node →right →left;
new_node →right ← node →right →right;
node →right →left →parent ← new_node;
node →right →right →parent ← new_node;
node →right →right ← new_node;
node →right →left ← node;
node →right →parent ← node →parent;
*parent_point ← node →right;
node →parent ← node →right;
allocate(another_new_node);
node →right ← another_new_node;
pointers[ ].update(node →parent, node →parent →right);
```

Algorithm 8: Right rotate case 2

3 Routings shared in both parts

Algorithm 9: Build Constant Tree

Algorithm 10: decode

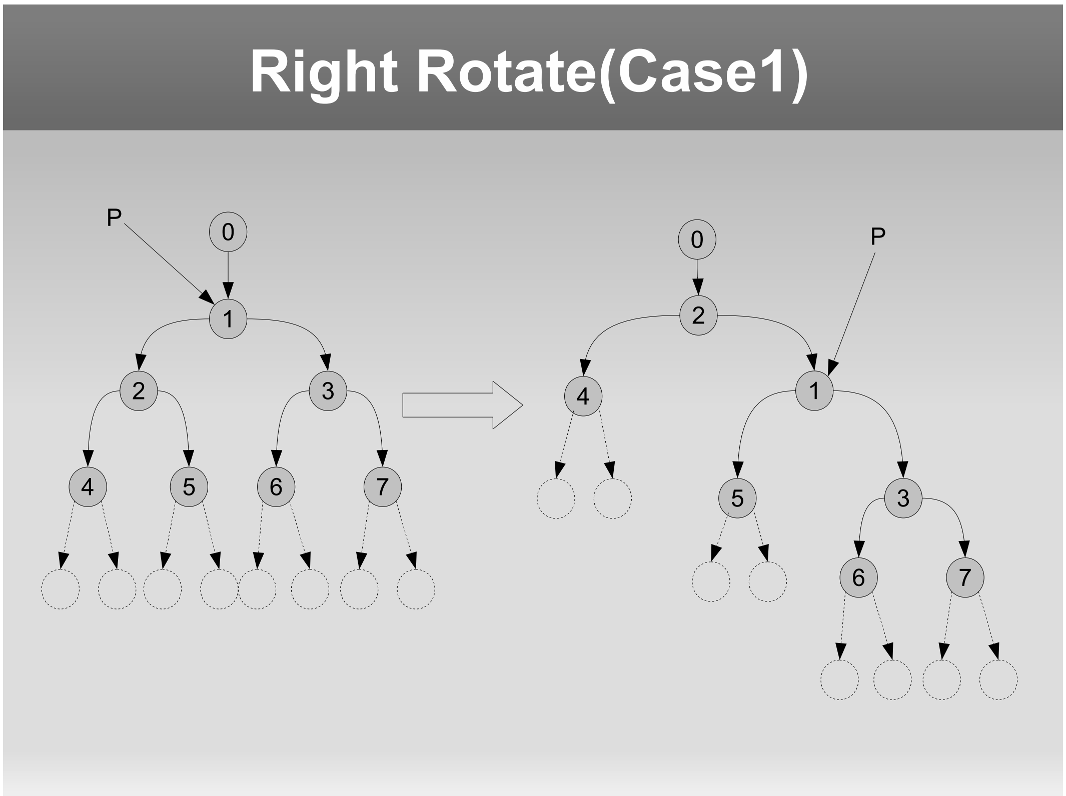

Figure 1: Right Rotate Case 1

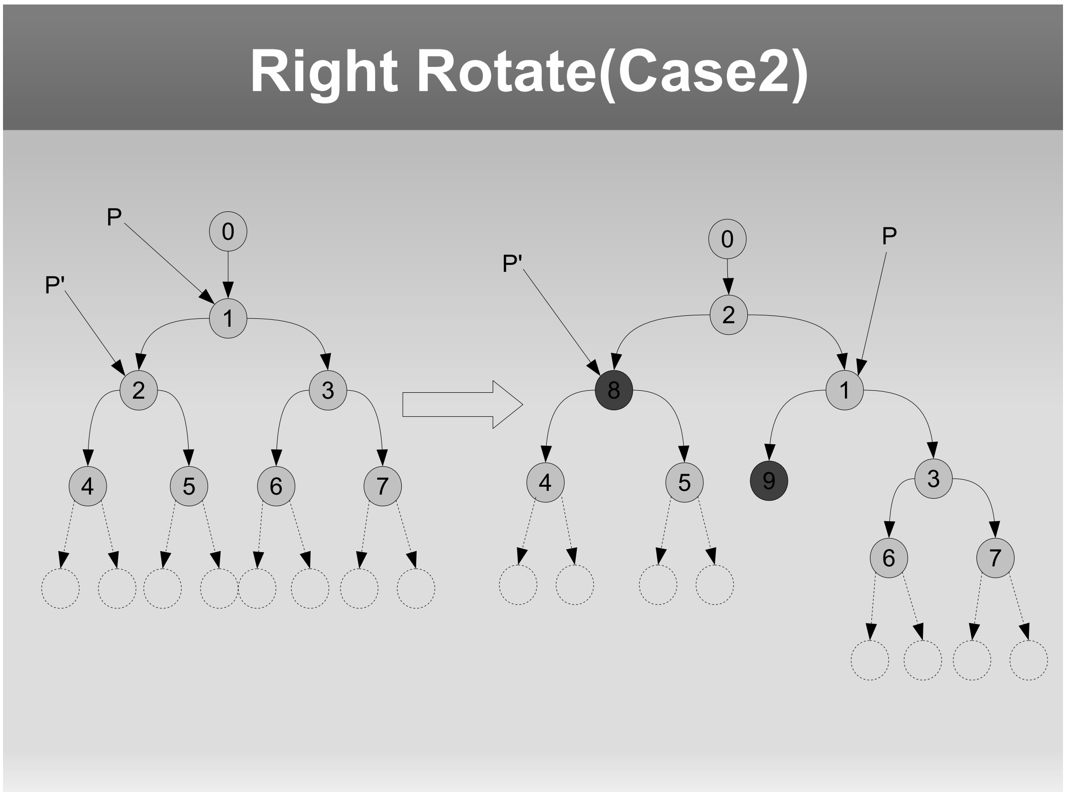

Figure 2: Right Rotate Case 2

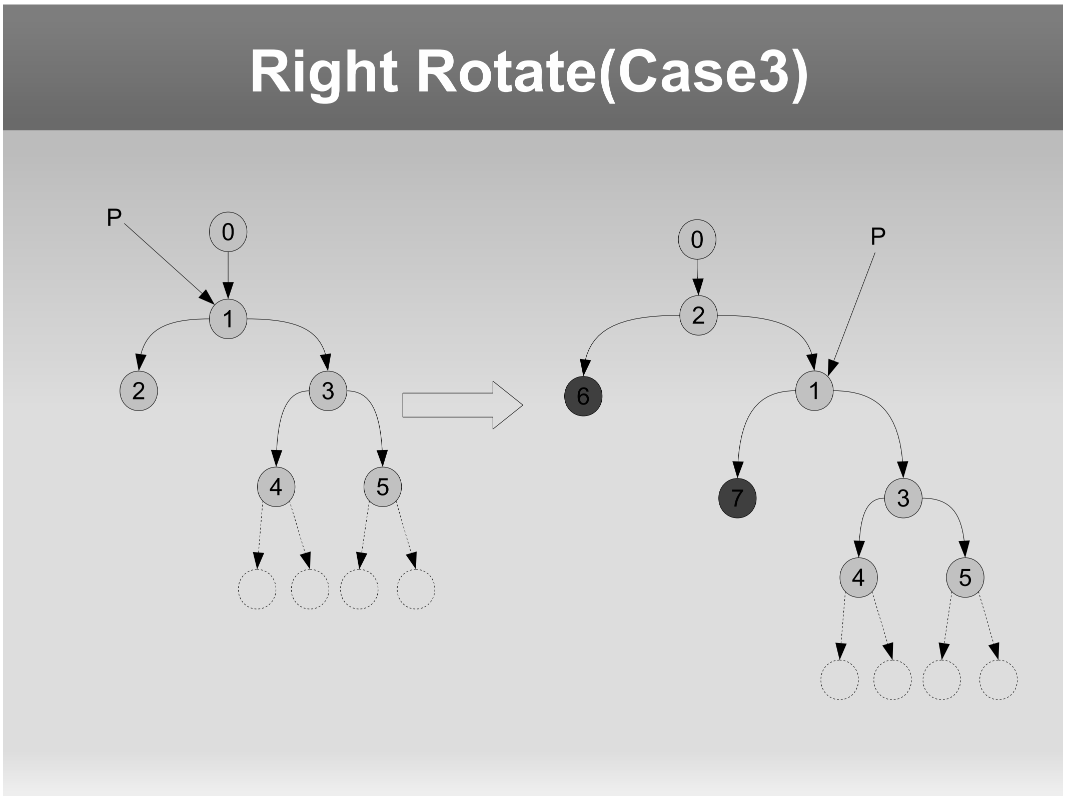

Figure 3: Right Rotate Case 3

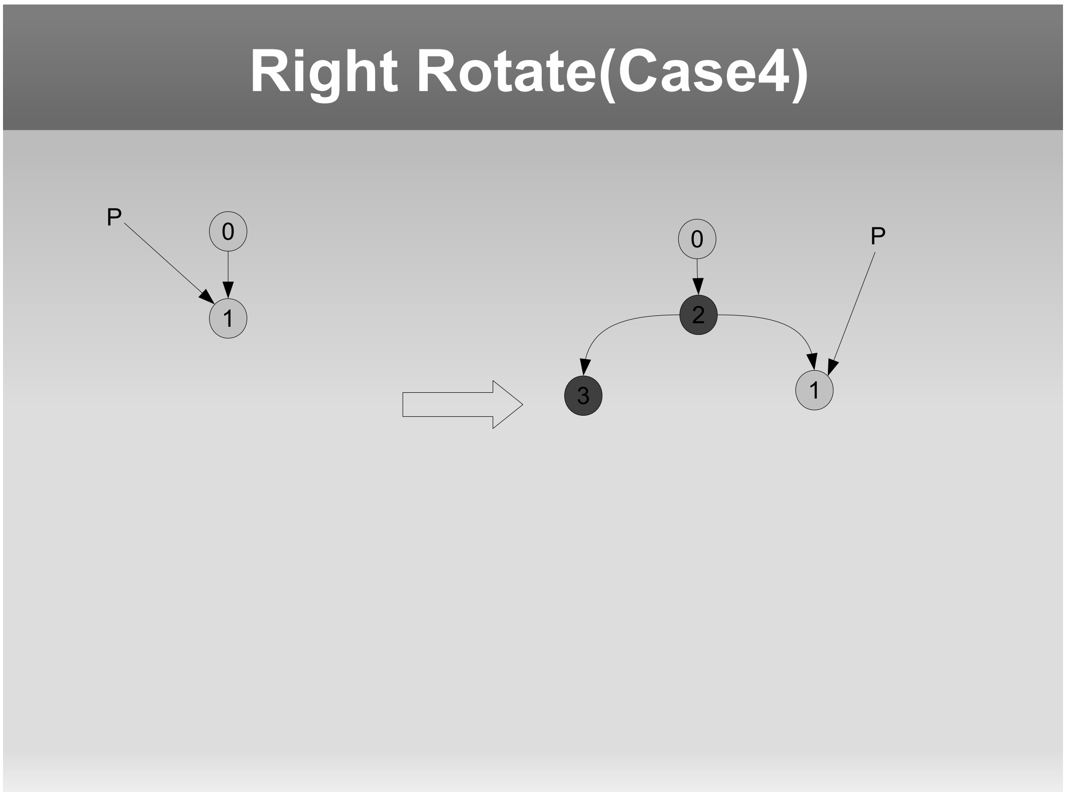

Figure 4: Right Rotate Case 4