Persistent Binary Tree Program

The persistent program in this report is a binary tree program written in C++. It imports and uses libraries in the Persistent Memory Development Kit (PMDK) to manage persistent memory. The libraries used in this program are libpmemobj and libpmemobj++. The program consists of two files: binaryTree.h and main.cpp, which uses structures and functions from binaryTree.h.

1) binaryTree.h

Three structures defined in binaryTree.h including their attributes and functions are as follows:

```
struct tree_node {
    pmem::obj::p<int> data;
    pmem::obj::persistent_ptr<tree_node> left, right, parent;
}
struct tree_iterator {
    pmem::obj::persistent_ptr<tree_node> currentNode;

    bool hasNext();                          // Return true if there is the next value. Otherwise, return false.
    bool hasPrevious();                      // Return true if there is the previous value. Otherwise, return false.
    int next();                              // Make currentNode point to the node containing the next value.
    int previous();                          // Make currentNode point to the node containing the previous value.
    void set_value(int value)                // Set the value in the node that currentNode points to.
}
struct binary_tree {
    pmem::obj::p<int> tree_size;
    pmem::obj::persistent_ptr<tree_node> tree_root;

    // Return an iterator pointing to the node that contains the minimum value in the tree.
    pmem::obj::persistent_ptr<tree_iterator> findMin(pmem::obj::pool_base &pop);
    pmem::obj::persistent_ptr<tree_iterator> findMin(pmem::obj::pool_base &pop, pmem::obj::persistent_ptr<tree_node> n);

    // Return an iterator pointing to the node that contains the maximum value in the tree.
    pmem::obj::persistent_ptr<tree_iterator> findMax(pmem::obj::pool_base &pop);
    pmem::obj::persistent_ptr<tree_iterator> findMax(pmem::obj::pool_base &pop, pmem::obj::persistent_ptr<tree_node> n);

    // Return true if the specified value is in the tree. Otherwise, return false.
    bool node_in_the_tree(pmem::obj::pool_base &pop, int v);

    // Return an iterator pointing to the node that contains the specified value if the value is in the tree. Otherwise, return null.
    pmem::obj::persistent_ptr<tree_iterator> find_node(pmem::obj::pool_base &pop, int v);
    pmem::obj::persistent_ptr<tree_iterator> find_node(pmem::obj::pool_base &pop, int v, pmem::obj::persistent_ptr
    <tree_node> n);

    // Insert a node if the specified value is not already in the tree. Otherwise, do nothing.
    pmem::obj::persistent_ptr<tree_node> insert_node(pmem::obj::pool_base &pop, int v);
    pmem::obj::persistent_ptr<tree_node> insert_node(pmem::obj::pool_base &pop, int v, pmem::obj::persistent_ptr
    <tree_node> n, pmem::obj::persistent_ptr<tree_node> parent);

    // Remove the node containing the specified value if that value is in the tree. Otherwise, throw an exception.
    pmem::obj::persistent_ptr<tree_node> remove_node(pmem::obj::pool_base &pop, int v);
    pmem::obj::persistent_ptr<tree_node> remove_node(pmem::obj::pool_base &pop, int v, pmem::obj::persistent_ptr
    <tree_node> n, pmem::obj::persistent_ptr<tree_node> parent);
}
```

NOTE: Functions in binary_tree are implemented using recursion.

2) main.cpp

- Firstly, main.cpp creates a memory pool at the location specified through a command line argument using **pmem::obj::pool<binary_tree>::create()**.
- Once the pool has been created, ten thousand nodes are inserted to the tree using **pmem::obj::persistent_ptr<tree_node>insert_node(pmem::obj::pool_base &pop, int v)**. The value of each node is chosen at random from integers between 0 – 100000 using **rand()**.
- After all the nodes have been inserted, **pmem::obj::persistent_ptr<tree_iterator> findMin(pmem::obj::pool_base &pop)** and **pmem::obj::persistent_ptr<tree_iterator> findMax(pmem::obj::pool_base &pop)** are called to find the maximum value and the minimum value, respectively.
- Finally, ten thousand integers between 0 - 100000 are chosen at random by **rand()**. For each value, **bool node_in_the_tree(pmem::obj::pool_base &pop, int v)** is called to check whether the value is in the tree or not. If the value is in the tree, the program removes the node that contains that value from the tree by using **pmem::obj::persistent_ptr<tree_node> remove_node(pmem::obj::pool_base &pop, int v)**.

Performance Analysis

For performance analysis, Gprof is used to breakdown the overheads of the program. The program spends 77.90 percent of its total execution time in **pmem::obj::persistent_ptr_base::operator=(pmem::obj::persistent_ptr_base&&)**, 6.82 percent in **void pmem::detail::create<tree_node>(pmem::detail::if_not_array<tree_node>::type*)**, 2.16 percent in **void pmem::detail::conditional_add_to_tx<pmem::obj::persistent_ptr_base>(pmem::obj::persistent_ptr_base const*, unsigned long, unsigned long)**, and 13.12 percent in other functions. The top ten functions that the program spends most of its execution time in and their fields are shown in table 1.

function name	% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call
pmem::obj::persistent_ptr_base::operator=(pmem::obj::persistent_ptr_base&&)	77.90	4.68	4.68	210742	0.02	0.02
void pmem::detail::create<tree_node>(pmem::detail::if_not_array<tree_node>::type*)	6.82	5.09	0.41	9528	0.04	0.04
void pmem::detail::conditional_add_to_tx<pmem::obj::persistent_ptr_base>(pmem::obj::persistent_ptr_base const*, unsigned long, unsigned long)	2.16	5.22	0.13	222572	0.00	0.00
std::enable_if<std::__and_<std::__not_<std::__is_tuple_like<int> >, std::is_move_constructible<int>, std::is_move_assignable<int> >::value, void>::type std::swap<int>(int&, int&)	1.66	5.32	0.10	10691	0.01	0.01
binary_tree::insert_node(pmem::obj::pool_base&, int, pmem::obj::persistent_ptr<tree_node>, pmem::obj::persistent_ptr<tree_node>)::{lambda()#1}::operator()() const	1.33	5.40	0.08	165241	0.00	0.03
void pmem::obj::transaction::run<>(pmem::obj::pool_base&, std::function<void ()>)	1.00	5.46	0.06	192050	0.00	0.00
pmem::obj::persistent_ptr<tree_node>::operator=(pmem::obj::persistent_ptr<tree_node>&&)	0.83	5.51	0.05	209577	0.00	0.02
std::_Function_base::~_Function_base()	0.83	5.56	0.05	192050	0.00	0.00
binary_tree::insert_node(pmem::obj::pool_base&, int, pmem::obj::persistent_ptr<tree_node>, pmem::obj::persistent_ptr<tree_node>)	0.83	5.61	0.05	165241	0.00	0.00
std::_Function_base::_Base_manager<binary_tree::insert_node(pmem::obj::pool_base&, int, pmem::obj::persistent_ptr<tree_node>, pmem::obj::persistent_ptr<tree_node>)::{lambda()#1}>::_M_destroy(std::_Any_data&, std::integral_constant<bool, false>)	0.83	5.66	0.05	165241	0.00	0.00

Table 1: Functions that the program spends most of its total execution time in.