Merkle 树实现与 RFC6962 标准验证实验报告

1. 实验目的

本实验旨在实现符合 RFC6962 标准的 Merkle 树数据结构，并验证其核心功能：

构建大规模 Merkle 树（10 万叶子节点）

生成并验证存在性证明（Inclusion Proof）

生成并验证不存在性证明（Exclusion Proof）

验证 RFC6962 标准的关键特性实现

2. 实验环境

编程语言：Python 3.9

依赖库：hashlib, struct, collections

3. 实验内容与代码

3.1 数据结构定义

```
class Node:
    def __init__(self):
        self.hash = None
        self.left_index = 0
        self.right_index = 0
        self.left_child = None
        self.right_child = None
        self.parent = None
```

Node 类表示 Merkle 树中的节点

hash 存储节点的哈希值

left_index 和 right_index 表示节点覆盖的叶子范围

left_child 和 right_child 指向子节点

parent 指向父节点

3.2 Merkle 树核心类

```
class SortedMerkleTree:
```

```
    def __init__(self):
        self.root = None
        self.leaf_count = 0
        self.leaf_data = []
```

SortedMerkleTree 类实现排序的 Merkle 树

root 存储根节点

leaf_count 记录叶子节点数量

leaf_data 存储所有叶子节点的原始数据

3.3 哈希计算函数

```
def _hash_leaf(self, data):
    return hashlib.sha256(LEAF_PREFIX + data).digest()

def _hash_internal(self, left_hash, right_hash):
    return hashlib.sha256(INTERNAL_PREFIX + left_hash + right_hash).digest()
```

_hash_leaf 实现 RFC6962 的叶子节点哈希计算：H(0x00 || data)

_hash_internal 实现内部节点哈希计算：H(0x01 || left_hash || right_hash)

使用 SHA-256 作为哈希函数，符合 RFC6962 标准

3.4 树构建算法

```
def _build_tree(self, start, end):
    if start == end:
        node = Node()
        node.left_index = start
        node.right_index = end
        node.hash = self._hash_leaf(self.leaf_data[start])
        return node
```

```
        mid = (start + end) // 2
        left_child = self._build_tree(start, mid)
        right_child = self._build_tree(mid + 1, end)

        node = Node()
        node.left_index = start
        node.right_index = end
        node.left_child = left_child
        node.right_child = right_child
        left_child.parent = node
        right_child.parent = node
        node.hash = self._hash_internal(left_child.hash, right_child.hash)

        return node
```

递归构建平衡二叉树

叶子节点直接计算哈希

内部节点合并左右子节点哈希

时间复杂度：O(n)，空间复杂度：O(n)

3.5 存在性证明

```
def generate_inclusion_proof(self, leaf_index):
        path = []
        node = self._find_leaf_node(leaf_index)
        while node.parent:
            parent = node.parent
            if node == parent.left_child:
```

```
                path.append(parent.right_child.hash)
            else:
                path.append(parent.left_child.hash)
            node = parent

        return MerkleProof(leaf_index, path)

def verify_inclusion_proof(self, data, proof):
        current_hash = self._hash_leaf(data)
        current_index = proof.leaf_index

        for sibling_hash in proof.path:
            if current_index % 2 == 0:
                current_hash = self._hash_internal(current_hash, sibling_hash)
            else:
                current_hash = self._hash_internal(sibling_hash, current_hash)
            current_index //= 2

        return current_hash == self.root.hash
```

generate_inclusion_proof 生成从叶子到根的路径证明

verify_inclusion_proof 通过重构路径验证证明

3.6 不存在性证明

```
def generate_exclusion_proof(self, data):
        index = self._find_insert_position(data)

        if index < self.leaf_count and self.leaf_data[index] == data:
```

```
            return None

        left_proof = None
        right_proof = None

        if index > 0:
            left_proof = self.generate_inclusion_proof(index - 1)

        if index < self.leaf_count:
            right_proof = self.generate_inclusion_proof(index)

        return (left_proof, right_proof)

    def verify_exclusion_proof(self, data, proof):
        left_proof, right_proof = proof

        if left_proof:
            left_data = self.leaf_data[left_proof.leaf_index]
            if left_data >= data or not self.verify_inclusion_proof(left_data, left_proof):
                return False

        if right_proof:
            right_data = self.leaf_data[right_proof.leaf_index]
            if right_data <= data or not self.verify_inclusion_proof(right_data, right_proof):
                return False

        return True
```

generate_exclusion_proof 通过查找相邻叶子生成证明

verify_exclusion_proof 验证相邻叶子存在且满足排序关系

符合 RFC6962 对不存在性证明的要求

3.7 辅助函数

```
def _find_leaf_node(self, index):
    node = self.root
    start = 0
    end = self.leaf_count - 1

    while start != end:
        mid = (start + end) // 2
        if index <= mid:
            node = node.left_child
            end = mid
        else:
            node = node.right_child
            start = mid + 1

    return node

def _find_insert_position(self, data):
    low, high = 0, self.leaf_count
    while low < high:
        mid = (low + high) // 2
        if self.leaf_data[mid] < data:
            low = mid + 1
```

```
            else:
                high = mid
        return low
```

_find_leaf_node 通过二分查找定位叶子节点

_find_insert_position 使用二分查找确定数据插入位置

4. 实验结果

```
Merkle Root: 65ac7896535c0377735ee28cae0c111323cbdc3f4e2548d6f59720037540a599
Inclusion proof for 5: Valid
Exclusion proof for 100001: Invalid
Exclusion proof for 0: Valid

进程已结束，退出代码为 0
```

比对相关资料能知道基本正确