

山东大学网络空间安全学院
网络空间安全创新创业实践



Project 3 Merkle Tree实现

姓名：张麟康

学号：201900301107

1 原理及其具体实现

1.1 Merkle Tree具体实现

```
class MerkleTreeNode: # Merkle Tree 结点
    def __init__(self, node_id, hash_val, left_child, right_child, parent):
        self.node_id = node_id
        self.hash_val = hash_val
        self.left_child = left_child
        self.right_child = right_child
        self.parent = parent
```

```

class MerkleTree:# Merkle Tree 定义
    def __init__(self, n, leaves, layers, cntlayers):
        self.n = n
        self.layers = []
        self.leaves = []
        self.cntlayers = int(math.log(n,2)+1)

    def fill_leaf(self, MerkleTreeNode): # 叶填充
        self.leaves.append(MerkleTreeNode)

    def construct_layers(self): # 构造层
        i = 1
        for i in range(self.cntlayers):
            self.layers.append([])

    def fill_leaves(self): # 叶填充
        self.layers = self.layers + [self.leaves]

    def build_tree(self): # 创建树
        self.fill_leaves()
        self.construct_layers()

```

```

    def build_tree(self): # 创建树
        self.fill_leaves()
        self.construct_layers()
        #build_tree
        len_layer = int(self.n/2)
        ctr = self.n #结点标识计数器
        for i in range(1,self.cntlayers):
            if(len_layer==1):# 如果是根节点
                ctr = ctr +1 #结点标识计数器更新
                self.layers[i].append(MerkleTreeNode(ctr, 0, 0, 0, 0))
                hash_val1 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][0].hash_val))
                hash_val2 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][1].hash_val))
                temp = operator.xor(hash_val1,hash_val2)
                self.layers[i][0].hash_val = hashlib.sha256(str(temp).encode()).hexdigest()
                self.layers[i][0].left_child = self.layers[i-1][0].node_id
                self.layers[i][0].right_child = self.layers[i-1][1].node_id
                self.layers[i-1][0].parent = ctr
                self.layers[i-1][1].parent = ctr

```

```

else: # 如果是内部结点
    k=0
    rng = len(self.layers[i-1]) # 子层长度
    for j in range(len_layer):
        if k < rng:
            ctr = ctr +1 #结点标识计数器更新
            self.layers[i].append(MerkleTreeNode(ctr, 0, 0, 0, 0))
            hash_val1 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][k].hash_val))
            hash_val2 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][k+1].hash_val))
            temp = operator.xor(hash_val1,hash_val2)
            self.layers[i][j].hash_val = hashlib.sha256(str(temp).encode()).hexdigest()
            self.layers[i][j].left_child = self.layers[i-1][k].node_id
            self.layers[i][j].right_child = self.layers[i-1][k+1].node_id
            self.layers[i-1][k].parent = ctr
            self.layers[i-1][k+1].parent = ctr
            k = k+2
        len_layer = int(len_layer/2)

```

```

leaf_count = 7
MT = MerkleTree(leaf_count,[],[],0)
for i in range(leaf_count):
    MT.fill_leaf(MerkleTreeNode(i+1,hashlib.sha256(str(i).encode()).hexdigest(),0,0,0))
MT.build_tree()

```

```

PS F:\course-project-2022> python310/python.exe F:\course-project-2022\Project
Number of Leaves: 8
1 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9 0 0 9
2 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b 0 0 9
3 d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35 0 0 10
4 4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce 0 0 10
5 4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdaf8a 0 0 11
6 ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d 0 0 11
7 e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683 0 0 12
8 7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451 0 0 12
9 4156108c729571c1f9756b355c03dc1f13bbd6ae93bfb74c73b0714a3562a04d 1 2 13
10 7229193557dec89e738090c3d09c3151d4f4e3f3132161f40620b24642f00e81 3 4 13
11 6760eb9c924f32a5d646ce0f961943e502fabe9e889286780f7e40ffb4265a7c 5 6 14
12 fae894871c59ac7f921094318c551543ef654a527903b53c5f37bc2e1eece273 7 8 14
13 6505de232d2001eb931a9bd4a3c96e10d41f6eaa7e737cc7aa6b45c8c80261e7 9 10 15
14 aab86e127fe5afe697b8892f6aed79934ab6b76763b74d87ffce6bedfea1c55f 11 12 15
15 b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79 13 14 0

```

```

199958 5da33aea49cbde8ff67c8202f7d51852763f3e0da0d96d38e11c7e271adfb8ec 199919 199920 199977
199959 bca31f277c52f88216269f9a0ad626e0963f91c96f2a26579e7fe3f0d5427ad2 199921 199922 199978
199960 450911a0b6f0e9e80a7bdbba328b1709348c0f4275f1637c877f7347e05327d0 199923 199924 199978
199961 27af77b23a16206a495ab434c6bbe23fc4ba6a44533a75dc0b80292d9b5625f0 199925 199926 199979
199962 f81106c56483afaad94ff18ccbc58685bfbfefa1b44a7a66b023e08be6627e9e 199927 199928 199979
199963 52aeec37320db2e407cb22280ee363388c202e468c0a73ad0fd573862c90d14 199929 199930 199980
199964 24e6499c9a19648ab0d9285aee8d265fc1e5b7aff6dc1562b210fb5087457671 199931 199932 199980
199965 c1e22eb36dec05f27085e8110d7ab620be30683f67cf89c0ea0d78b85cf029c 199933 199934 199981
199966 5176978234fc0cfcfc1a1f5d6248be58ac6d43dc308f4aab80360f19c03bbef 199935 199936 199981
199967 fcac483ccda5665ce6c2cec138d795109f2116aa851a2a8214dbd96b9b853660 199937 199938 199982
199968 48c08e966d3a7b8b4a68eb8a26d2ad12e32d81c017b2b094418bfc57ad54cb8c 199939 199940 199982
199969 8b63ecfe027c1f8a7eff198208e5907a6e20eba58dbf90a70aad1fe550744169 199941 199942 199983
199970 409b67547c6606e4773d886f8d210adf6fb813f606b21eb564d593694bb0e2fa 199943 199944 199983
199971 b3dbbc5a25635ee94595d38cd5d0153b58d762631d1e8e742aec6eb30d198641 199945 199946 199984
199972 50638b3169f95ac3ee28ff2efc24a8949922bbace6a6d39ac5751e1502db7364 199947 199948 199984
199973 44a4d4dcec3ee9c452b8e9cef8d0a517bb230f9b6ea2708cbb4b0747a6c89f 199949 199950 199985
199974 f94cef69ecbaf222fbcce5b290e4172273e2646e227ba6913099af9f9d510dda 199951 199952 199985
199975 e0f2a47db7c71c242f3c7e9283545159163cc896162db79277695c394f45310e 199953 199954 199986
199976 a7fbc9f46f9c9dd6a2f5f12a1020f50589010ed22513d5d5f9e8bb2fc99fdd54 199955 199956 199986
199977 c93fc63c9b47a3e372c803d2b93b0ad889fd64c7ba1a65876c98714940b081b0 199957 199958 199987
199978 725d69c47056aec0d3054d73ca2f1e0f31d6f6dd780cb967cceedb97f28eded5 199959 199960 199987
199979 dcae3e3aee1a69edf606e3a60643438d9577c7043c49d3bee25890ec12ad091 199961 199962 199988
199980 b319fb34f03596a96af0c0b83a96e556bea537b039cd259e09850f344920dbe 199963 199964 199988
199981 6a0747319a4ac847f40da475e2d41cebe8c05d88b9dee1c79b155b076c5e369c 199965 199966 199989
199982 ff375d27cd2d5e5016ba2663fef3c6cf82b547af178cd2eb1cc4c5f3672a5f51 199967 199968 199989
199983 cffb990b92ba7d51ac6672deac01813eca75995d91442bed877de03731658a2c 199969 199970 199990
199984 438289236f984fff1f873e7c12ffee6be9910d2f58696b173a27832ba2b5ce43 199971 199972 199990
199985 9e5b785dad7d966cdf227abdec99f2929728245ce301acbdef08155e70f718ba 199973 199974 199991
199986 e44f76b81d656bb5fb6a73ed3826cbb7c7e8b43210d12884d63d26ba3d064297 199975 199976 199991
199987 0c165fe2b3ae417a702d90837c6a2bc561b5764a437dabe9b21d91dd3a51a0dd 199977 199978 199992
199988 5e978097f98e4c788e95d058254ffb8a679782ecae139369b0c17c58f8b4f26 199979 199980 199992
199989 b018d88d4ccc2d8974cc51f675698740ff22fc4d10f8cf93c9239e7eb5abbf8b 199981 199982 199993
199990 2748670198eaaedb307e8bfd75933fd57e12cbac97cd7ea85749481b6ed12772 199983 199984 199993
199991 231cc7fbdf86c1213b07fa25c3af206b143369fa19fc8caf33b4d384c5925d8b 199985 199986 199994

```

1.2 对特定元素的存在性证明

```

def find_node_from_id(self, node_id):
    try:
        for i in range(self.cntlayers):
            for j in range(len(self.layers[i])):
                temp = self.layers[i][j].node_id
                if (temp == node_id):
                    return self.layers[i][j]
    except:
        print("node does not exist in the tree.")

```

```

def find_path(self, node_id):
    try:
        path = []
        pivot = self.find_node_from_id(node_id)
        for i in range(self.cntlayers):
            parent = 0
            for j in range(len(self.layers[i])):
                if(pivot.parent == self.layers[i][j].node_id):
                    parent = self.layers[i][j]
                    if parent.left_child == pivot.node_id:
                        path.append(self.find_node_from_id(parent.right_child).hash_val)
                    else:
                        path.append(self.find_node_from_id(parent.left_child).hash_val)
                    pivot = parent
            return path
    except:
        print("node does not exist in the tree.")

```

```

TypeError: can only concatenate str (not "int") to str
PS F:\course-project-2022> & D:/Python310/python.exe "f:/course-project-2022/Project 3 Merkle Tree/merkle_tree.py"
Number of Leaves: 8
1 5feceb66ffcc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9 0 0 9
2 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b 0 0 9
3 d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35 0 0 10
4 4e07408562bedb8b0ce05c1decfe3ad16b72230967de01f640b7e4729b49fce 0 0 10
5 4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdafb8a 0 0 11
6 ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d 0 0 11
7 e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683 0 0 12
8 7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451 0 0 12
9 4156108c729571c1f9756b355c03dc1f13bbd6ae93bfb74c73b0714a3562a04d 1 2 13
10 7229193557dec89e738090c3d09c3151d4f4e3f3132161f40620b24642f00e81 3 4 13
11 6760eb9c924f32a5d646ce0f961943e502fabe9e889286780f7e40ff4265a7c 5 6 14
12 fae894871c59ac7f921094318c551543ef654a527903b53c5f37bc2e1eece273 7 8 14
13 6505de232d2001eb931a9bd4a3c96e10d41f6eaa7e737cc7aa6b45c8c80261e7 9 10 15
14 aab86e127fe5afe697b8892f6aed79934ab6b76763b74d87ffce6bedfea1c55f 11 12 15
15 b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79 13 14 0
Root of the Merkle tree:
b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79
=====inclusion=====
Hash Value of the Node with ID 6:
ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
Path of the Node with ID 6:
4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdafb8a
fae894871c59ac7f921094318c551543ef654a527903b53c5f37bc2e1eece273
6505de232d2001eb931a9bd4a3c96e10d41f6eaa7e737cc7aa6b45c8c80261e7
Found root with Path and Hash of the Node with ID 6:
b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79
=====exclusion=====
node 9999 does not exist in the tree.
PS F:\course-project-2022> 

```

1.3 对特定元素的不存在证明

```

print('====exclusion=====')
node_id=9999
try:
    node_hash = MT.find_node_from_id(node_id).hash_val
    path = MT.find_path(node_id)
    found_root = root_from_path(path,node_hash)
    print("Hash Value of the Node with ID "+str(node_id)+": ")
    print(node_hash)
    print("Path of the Node with ID "+str(node_id)+":")
    for i in path:
        print(i)
    print("Found root with Path and Hash of the Node with ID "+str(node_id)+": ")
    print(found_root)
except:
    print("node "+str(node_id)+" does not exist in the tree.")

```

```

TypeError: can only concatenate str (not "int") to str
PS F:\course-project-2022> & D:/Python310/python.exe "f:/course-project-2022/Project 3 Merkle Tree/merkle_tree.py"
Number of Leaves: 8
1 5feceb66ffcc8d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9 0 0 9
2 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b 0 0 9
3 d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35 0 0 10
4 4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce 0 0 10
5 4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdafb8a 0 0 11
6 ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d 0 0 11
7 e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683 0 0 12
8 7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451 0 0 12
9 4156108c729571c1f9756b355c03dc1f13bbd6ae93bfb74c73b0714a3562a04d 1 2 13
10 7229193557dec89e738090c3d09c3151d4f4e3f3132161f40620b24642f00e81 3 4 13
11 6760eb9c924f32a5d646ce0f961943e502fabe9e889286780f7e40ffb4265a7c 5 6 14
12 fae894871c59ac7f921094318c551543ef654a527903b53c5f37bc2e1eece273 7 8 14
13 6505de232d001eb931a9bd4a3c96e10d41f6eaa7e737cc7aa6b45c8c80261e7 9 10 15
14 aab86e127fe5afe697b8892f6aed79934ab6b76763b74d87ffce6bedfea1c55f 11 12 15
15 b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79 13 14 0
Root of the Merkle tree:
b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79
=====inclusion=====
Hash Value of the Node with ID 6:
ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
Path of the Node with ID 6:
4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdafb8a
fae894871c59ac7f921094318c551543ef654a527903b53c5f37bc2e1eece273
6505de232d001eb931a9bd4a3c96e10d41f6eaa7e737cc7aa6b45c8c80261e7
Found root with Path and Hash of the Node with ID 6:
b057d9cb0b62122cf3c28e1141033e8ed655d5449c5d272357f81d34553c1e79
=====exclusion=====
node 9999 does not exist in the tree.
PS F:\course-project-2022> 

```