Experimental report——Merkle Tree following RFC6962

姓名：李祥方

学号：201900460041

完成时间：6月30日

1 前置知识

$MerkleTree$：Merkle Tree可以看做Hash List的泛化（Hash List可以看作一种特殊的Merkle Tree，即树高为2的多叉Merkle Tree). 在最底层，和哈希列表一样，我们把数据分成小的数据块，有相应地哈希和它对应。但是往上走，并不是直接去运算根哈希，而是把相邻的两个哈希合并成一个字符串，然后运算这个字符串的哈希，这样每两个哈希就结婚生子，得到了一个"子哈希"。如果最底层的哈希总数是单数，那到最后必然出现一个单身哈希，这种情况就直接对它进行哈希运算，所以也能得到它的子哈希。于是往上推，依然是一样的方式，可以得到数目更少的新一级哈希，最终必然形成一棵倒挂的树，到了树根的这个位置，这一代就剩下一个根哈希了，我们把它叫做 Merkle Root.

2 实验过程

根据以上前置知识，编写了以下的代码，其中主要包括随机生成数据，创造Merkle树和验证函数:

```python
#第一步随机生成数据
def data(l):
    List = [''.join(random.sample(string.digits+string.ascii_letters, 5)) for _ in range(0, l)]
    return List

#第二步生成一个Merkel树
def create(data,LEN):
    depth = math.ceil(math.log(LEN, 2)+1)
    #创建一个二维数组
    merkletree = [[]]
    #二维数组倒置
    merkletree[0] = [(hashlib.sha256(i.encode())).hexdigest() for i in data]

    for i in range(1, depth):
        LEN = int(len(merkletree[i-1])/2)
        node=[(hashlib.sha256(merkletree[i-1][2*j].encode() + merkletree[i-1][2*j+1].encode())).hexdigest() fo
        merkletree.append(node)
        if len(merkletree[i-1])%2 == 1:
            merkletree[i].append(merkletree[i-1][-1])
    return merkletree
```

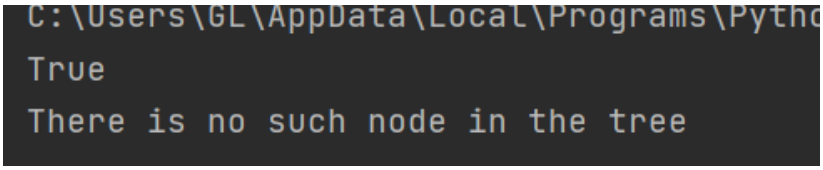
```python
    hash = (hashlib.sha256(m.encode())).hexdigest()
    if hash in tree[0]:
        node = tree[0].index(hash)
    else:
        return "There is no such node in the tree"
    #记录查验路线
    path = []
    for d in range(0,depth):
        if node%2==0:
            if node != len(tree[d]) - 1:
                path.append([tree[d][node], tree[d][node + 1]])
        else:
            path.append([tree[d][node-1], tree[d][node]])
        node = int(node/2)
    path.append([tree[-1][0]])
    if hash != path[0][0] and hash != path[0][1] and path[-1][0] != tree[-1][0]:
        return False
    depth = len(path)
    for i in range(0, depth - 1):
        node = (hashlib.sha256(path[i][0].encode() + path[i][1].encode())).hexdigest()
        if node != path[i + 1][0] and node != path[i + 1][1]:
            return False
    if (hashlib.sha256(path[-2][0].encode() + path[-2][1].encode())).hexdigest() != path[-1][0]:
        return False
    return True
```

3 实验结果

测试部分代码：

```python
#以下是测试结果：
LEN=100000
#生成数据集合
d=data(LEN)
#生成树
tree=create(d,LEN)
#打印树（结构倒置）
#print(tree)
print(verification(d[random.randint(0,99999)],tree))
print(verification('1234',tree))
```

测试结果：



由测试结果可知,函数的编写是正确的.