Merkle Tree

1.具体的项目代码说明

首先建立一个Merkel tree ， 基于Merkel tree 的基本原则 构建一颗Merkel tree。

构建node类：节点的属性有左孩子右孩子 hash值 数据 父节点指针 两个标志位

```
    def __init__(self, item):   #节点的内容
        self.item = item

        self.left = None
        self.right = None

        self.hash = None
        self.data = None

        self.parent=None    #指向父节点
        self.flag=0  #如果flag不是0那么就是叶子节点
        self.time=0
```

为了方便数据输入定义一个产生value的函数

```
generate_value(i):
```

使用hashlib让hash值稳定。

```
def Hash_single(string):
    s = hashlib.sha256()
    s.update(string.encode())
    b = s.hexdigest()
    return b
```

构建一个Merkel树类用于构建树，构建方法把两个节点进行hash然后把hash值赋值给父节点，若为奇数个节点

```
def merkle(self,elelist):
```

用traversse函数来遍历输入所有的节点，输出顺序是层序遍历。

```
def traverse(root_node):  # 层次遍历
```

用于把中间用于输入的第三方列表刷新

```
def flesh(root_obj):
```

POP函数，用于返回一个节点的关键路径，关键路径用于重构根hash

```
def POP(root_obj,id):  #返回关键路径
```

POA函数，不存在证明 用于返回这个节点的前一个节点和后一个节点的关键路径，用夹逼的方法来确定出来这个不存在

```
def POA(root_obj,id):
```

重构Merkle树

```
def reconstruct(Input,KeyPath):
```

对于同一层的非叶子节点的数目如果是奇数，那么把这个节点的hash直接给父节点。如果叶子节点存在奇数就把最后一个叶子节点复制一个然后构成偶数个叶子节点。

运行指导

直接运行即可。

代码运行过程截图

首先层次遍历一下所有的节点：（为了方便顺便把属性也打印出来，parent为父节点也就是非叶子节点叶子节点用序号表示，因为是在插入前排序 所以这里相当于排序了，方便POA的时候使用）

下面的截图是Merkel tree重构的时候产生的父节点和hash 可以和上面的进行对比发现是一样的

```
b1fdb7e6484a09ed9aeca833894837de9e5345782bfa52e15510d68671df7b9b
b9af57f9acb9a88fc50e5ddbc672b0f844b1c46fc097cab978f6bd7464512e1a
3e0cb2f34371c6fd3f8f89d1e6f1ecdc48150d4dc5afaf57eae568ef8021a0b5
4b5a005559b6e3d0ac0a7668fab09dcf2247a6ac68d043adf62aa69e3da2ad4d
2a9f460c300da049ae73519c2034950037b52fb75f66f25db5f86e0cf8f3789a
```

这个是POP返回的关键路径用于重构hash

```
关键路径为 ['3a161a43c209d461c3f61739ccfec0d1722fdb5de31b49802102b3c1aa484a8d', '6ba167ff3d76976bd3f4da07b02e2ada8687a01cc3f24f23fc5d28f6049ed608',
 '717d1e6f22be84a11dd5a94598c667088da00235f1dd7ccab24d1fee8a2ebbd9', 'a7bfc8eb9beb2b34c61936d5cc20b58704b6125f66cc54a683c2d638ab1ce627',
 '2a9f460c300da049ae73519c2034950037b52fb75f66f25db5f86e0cf8f3789a']
```

重构完毕如果发现重构出来的根hash 和本地保存的Merkel的根hash相同

```
验证成功
```

下面返回的是不存在性证明的两个关键路径 同样的在进行两个重构也能检验，因为POA相当于两个POP那么这里就不重复了。

```
不存在性证明关键路径1: ['b1fdb7e6484a09ed9aeca833894837de9e5345782bfa52e15510d68671df7b9b', '6ba167ff3d76976bd3f4da07b02e2ada8687a01cc3f24f23fc5d28f6049ed608',
 '717d1e6f22be84a11dd5a94598c667088da00235f1dd7ccab24d1fee8a2ebbd9', 'a7bfc8eb9beb2b34c61936d5cc20b58704b6125f66cc54a683c2d638ab1ce627',
 '2a9f460c300da049ae73519c2034950037b52fb75f66f25db5f86e0cf8f3789a']
不存在性证明关键路径2: ['3a161a43c209d461c3f61739ccfec0d1722fdb5de31b49802102b3c1aa484a8d', '6ba167ff3d76976bd3f4da07b02e2ada8687a01cc3f24f23fc5d28f6049ed608',
 '717d1e6f22be84a11dd5a94598c667088da00235f1dd7ccab24d1fee8a2ebbd9', 'a7bfc8eb9beb2b34c61936d5cc20b58704b6125f66cc54a683c2d638ab1ce627',
 '2a9f460c300da049ae73519c2034950037b52fb75f66f25db5f86e0cf8f3789a']
```