项目说明

在SM3代码库中

✅Project: implement Merkle Tree following RFC6962

运行说明

开发环境：Windows WSL（Ubuntu18.04）

默认执行环境：Linux/Windows

运行方式：在linux下，直接运行build/中的MerkleTree可执行文件，build/中也有VS编译的exe文件

`$: ./build/MerkleTree` 或 双击 MerkleTree_vs.exe

文件说明

- build/ 里面有Makefile和可执行文件；
- build/MerkleTree ELF可执行文件，linux下执行；
- build/MerkleTree_vs.exe PE可执行文件，数据量比较大的时候测试；
- build/MerkleTree_debug.exe PE可执行文件，测试量小可以看到很多打印信息；
- main.cpp 函数中有测试代码，可以调整测试的数据量，数据量较小时，可以将MerkleTree.h中的`#define DEBUG`解除注释，可以看到完整的代码流程；
- MerkleTree.h MerkleTree的类声明；
- MerkleTree.cpp MerkleTree的类函数定义；
- mySM3.h 使用的之前写的SM3作为hash函数；
- mySM3.cpp 使用的之前写的SM3作为hash函数

代码运行过程：

- **示例文件：**MerkleTree_debug.exe
- **环境： (Windows VS)**
- **参数说明**：减少了banchsize，添加了DEBUG的宏，方便打印东西

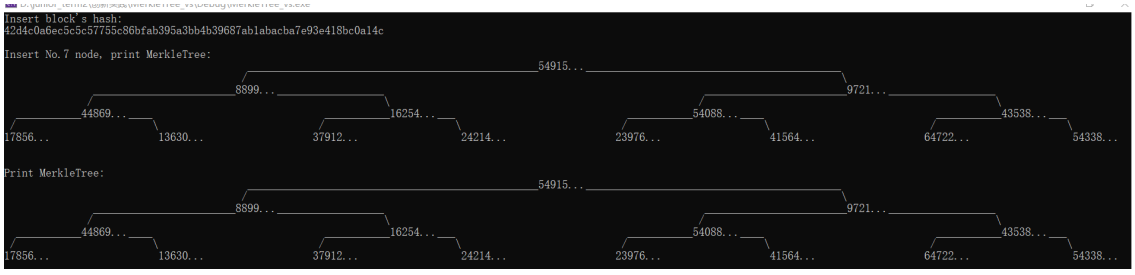
- **运行截图1：**

```
D:\junior_term2\创新实践\MerkleTree_vs\Debug\MerkleTree_vs.exe
        MerkleTree written by zzp

Data Block Size = 1024 byte
Banch Size Size = 8
0M byte data in total
Set DEBUG can print the whole MerkleTree(adjust in a small banchsize <=8)

Insert block's hash:
c04533e2d946d5903deb702eeda9ab360a4b674dc66603901dfd5166446a180a

Insert No.0 node, print MerkleTree:
17856...


Insert block's hash:
a85df36d6b4dbea01d4b102fa6cd9fb929cb1cd68de94e0c56552e3ea659d1d8

Insert No.1 node, print MerkleTree:
  ___________31085..._____
 /                        \
17856...                  23976...


Insert block's hash:
1894144b4fb380c27e35190808293af617edbce053f8bdac25f9fca717a93300

Insert No.2 node, print MerkleTree:
                          ____________62102..._____
                         /                         \
            ___________59258..._____                23976...
           /                        \
17856...                  37912...


Insert block's hash:
d2fc8fd94b1c09baa41e9992a2fabfcc0524929d4608a1b401546eee1ab98877

Insert No.3 node, print MerkleTree:
                          ____________36344..._________________
                         /                                     \
            ___________59258..._____                           5565..._____
           /                        \                         /            \
17856...                  37912...                23976...                64722...
```

- **截图解释**:

开头打印出了一些基本代码信息，包括预设的数据块大小，测试的数据块个数等，随后main函数调用randtest函数，随机生成数字充当我们的测试数据，随后new一个MerklTree类对象，动态建立MerkleTree，图中可以明显看出更新的方式，是**所有的叶子节点从左向右接收新节点。**

- **运行截图2:**



- **截图解释:**

插入全部8个节点（编号0-7）后，打印完全形态的二叉树

- **运行截图3:**

```
verify Block[4]
verify's hash:
3e35cc7c5b7f3974c458b8efdeb20be718299688bcbdc89f02f140020eb2023d

SM3 input:
c04533e2d946d5903deb702eeda9ab360a4b674dc66603901dfd5166446a180a3e35cc7c5b7f3974c458b8efdeb20be718299688bcbdc89f02f140020eb2023d
45af3dd706b4ece9d61eb528a94d32ac20541381a5350feafb764b0236f24755
45af3dd706b4ece9d61eb528a94d32ac20541381a5350feafb764b0236f24755
SM3 input:
45af3dd706b4ece9d61eb528a94d32ac20541381a5350feafb764b0236f247557e3f2c34dec5a2f200960bd408554ce57afb7d64db114057eb3f0359b15eacc7
c322fa8ef3bf41f6219f3664aba4f459b49f81e1b221d4dc64c40f2e56d5a63c
c322fa8ef3bf41f6219f3664aba4f459b49f81e1b221d4dc64c40f2e56d5a63c
SM3 input:
c322fa8ef3bf41f6219f3664aba4f459b49f81e1b221d4dc64c40f2e56d5a63cf9252bfb141831f62adc196e79e991a14b0a540de92027a0159848b4d7b058f2
83d6519efbb1629967faaf507b13fbae18c21fe171ef1824541f0444b7e90949
83d6519efbb1629967faaf507b13fbae18c21fe171ef1824541f0444b7e90949
Success! This datacblock is in the MerkleTree

memset Block[4][0] - Block[4][512] = 0,and verify:
verify's hash:
62edd1c6c4542572f68a2a687af44e5cd809a5a72cacaa00b07b38581f23e515

Failure! This datacblock isn't in the MerkleTree

Build a MerkleTree including 8 blocks and verify two blocks costs 0.126s
```

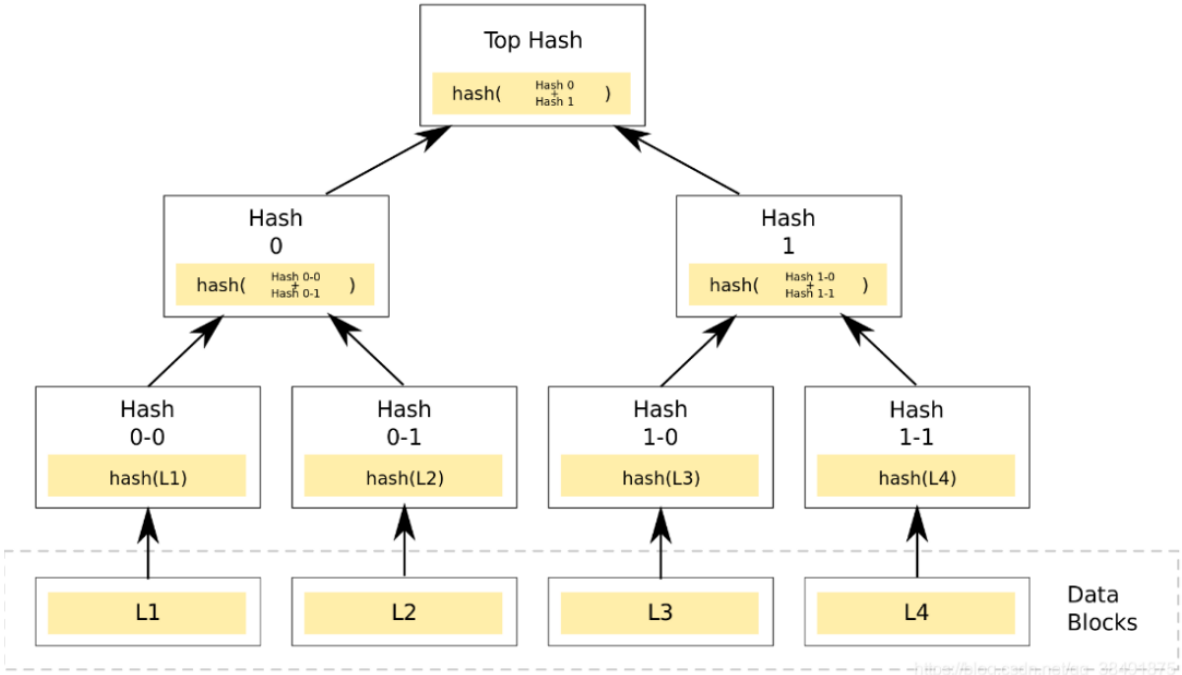
- **截图解释：**

  主要看验证部分，我们验证了一下Block[4]，先遍历叶节点，确定Block[4]的hash在不在叶节点中，如果在的话，再调用函数获得他的验证路径，RFC6962中有对验证路径的说明。

  然后再修改一下Block[4]的数据，将其前一半置0，然后再去verify一下，可以在截图中看到，这次的验证是Failure！

原理：

Merkle树看起来非常像二叉树，其叶子节点上的值通常为数据块的哈希值，而非叶子节点上的值

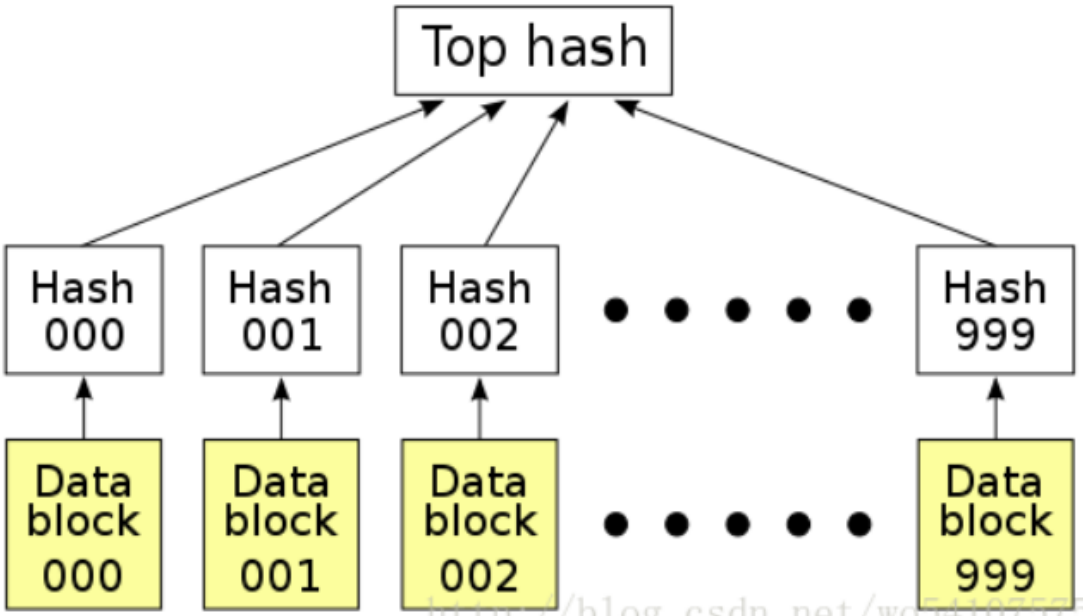


在构造Merkle树时，首先要对数据块计算哈希值，通常，选用SHA-256等哈希算法。但如果仅仅防止数据不是蓄意的损坏或篡改，可以改用一些安全性低但效率高的校验和算法，如CRC。然后将数据块计算的哈希值两两配对（如果是奇数个数，最后一个自己与自己配对），计算上一层哈希，再重复这个步骤，一直到计算出根哈希值。

Hash List

在点对点网络中作数据传输的时候，会同时从多个机器上下载数据，为了校验数据的完整性，更好的办法是把大的文件分割成小的数据块（例如，把分割成2K为单位的数据块）。如果小块数据在传输过程中损坏了，那么只要重新下载这一快数据就行了，不用重新下载整个文件。

BT下载的时候，在下载真正数据之前，我们会先下载一个Hash List。同时，我们把每个小块数据的Hash值拼到一起，然后对这个长字符串再作一次Hash运算，这样就得到Hash列表的根Hash(Top Hash or Root Hash)。下载数据的时候，首先从可信的数据源得到正确的根Hash，使用root hash校验hash list 然后使用hash list 校验数据块。



Merkle Tree

merkle tree 是一种泛化的hash list，hash list 可以看做树高为2的多叉merkle tree。有n个数据块，计算所有数据块hash值，然后相邻hash值再合并计算hash，直到计算到根hash，这样树高就为log(n)。

merkle tree的检索也就是对树的二分查找，时间是O(logn)，merkle tree的更新、插入和删除没有同一的标准，根据使用场景的不同，可以设计灵活的方法，既可以使用一些平衡树的方法（AVL树、红黑树等）也可以强行加树高，这在不同的场景下会有不同的应用。