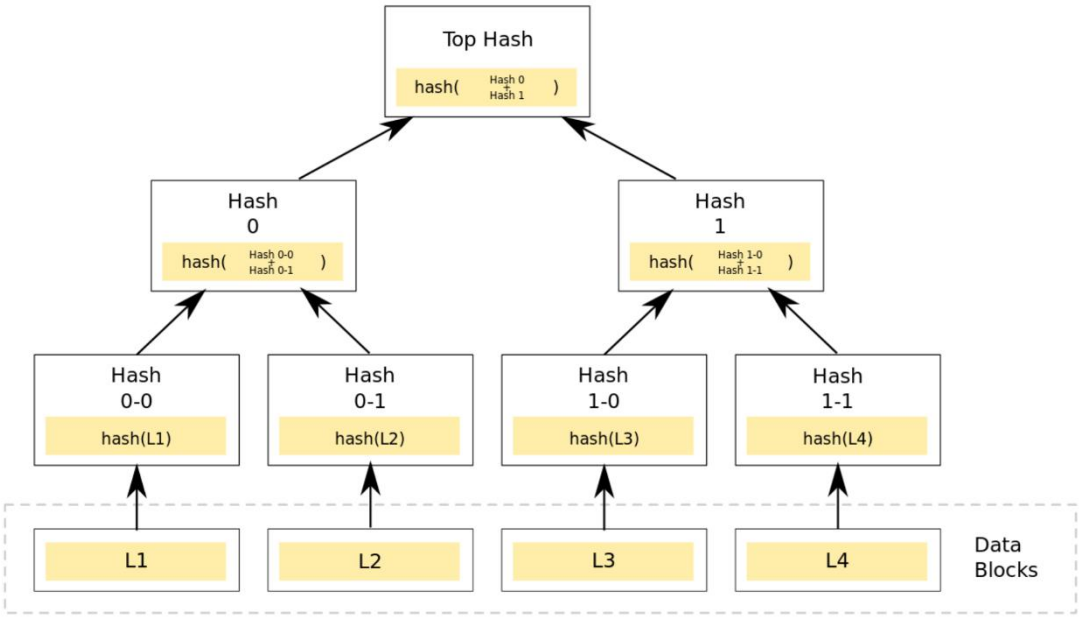
Merkle 树的实现与（不）存在性证明

一、 Merkle 树的实现



Merkle 树的实现原理可以由上图简单概括，即将数据分块先分别进行 HASH，然后自底而上，每两个分块进行合并和再 HASH，直到只剩一个 HASH 节点为止，最后的 HASH 节点即为 Merkle 树的根。在比特币的 Merkle 树中两次使用到了 SHA256 算法，因此其加密哈希算法也被称为 double-SHA256。即上述的每一步的 HASH 中，实际上进行了两次 SHA256 算法。

当数据块的个数是偶数时，可以很好的进行上述步骤，但是当数据块的个数为奇数时，一种处理方法是将最后一个分块复制一份，变成偶数个数据块。但是若有一个恶意的攻击者产生了两个相同的数据分块时，就很难分清楚最后一个数据块是真正的数据还是复制的分块。针对于这个问题，一种处理方法是认为一般情况不会出现相同数据块的情形，若出现了两个相同数据块，则认为是恶意的；即认为最后一块是为了变成偶数个而复制所得的。

另一种处理方法是，并不执着于完美二叉树，而是实现满二叉树即可。即当数据块的个数为奇数时，将最后的一个数据块提到上一层中。而当数据块的个数为偶数时，处理方法如常。如此，奇数是数据块所在的层数至多相差 1。对

于这样的二叉树的节点来说，要么没有孩子（叶子节点），要么有两个孩子。

此时，若叶子节点的个数为 n，根据树的所有节点的总出度等于总入度，可以得到内部节点的个数为 n-1，即这个树的总结点为 2*n-1。

对于这样的数据结构，可以将其依次放进一个数组中，此时如果一个内部节点的索引为 i，则它的两个孩子的索引分别是 2*i+1 与 2*i+2。

我第一个实现思路是按照上述的方式，先根据数据的个数，确定其在二叉树数组中的索引位置，然后再根据父节点和子节点之间的索引关系，通过合并再 HASH 的方式，自下而上构建一个二叉树。如此可以求得最后的 Merkle 树根。

具体实现方式见“Merkle 树实现（初步）”

```
叶子节点的个数为:
4
请从左到右输入叶子节点的数据：
daesfe
dgave
sfvetrn
atbyhnsr
建立merkle树
先序遍历merkle树:
689A5A282ADAAC9E200A4D095C6A514A7B7CDD29FA502A3B232B607E3E7BFE11
5D708F82936C4AFFCA68DC254E3A5A8C7C6A8C444E095B8F0D3FCE942B4C8F8C
307D3491575CEF8E4E1EDF3C553D5B0F38803A5210EA799A6C9AAB6A575D805F
671A469D5F6C270D4ABC20793F5A93990B9FADBA0B175C157A734154587451EF
3F50553B9A111F5072FD71BE2F919E452592778E3D8A6C128A9323106C4E8D9E
EC87906FDA8B96486C2B829E931A3EFE5A3B2D9D4F720D73810E6965FF6B515B
2D9E4CAB7AED7A24AC329C422D5C432B265D778849AC3A7512CD3BAE2E3C1B11

树根为：
689A5A282ADAAC9E200A4D095C6A514A7B7CDD29FA502A3B232B607E3E7BFE11
请按任意键继续. . .
```

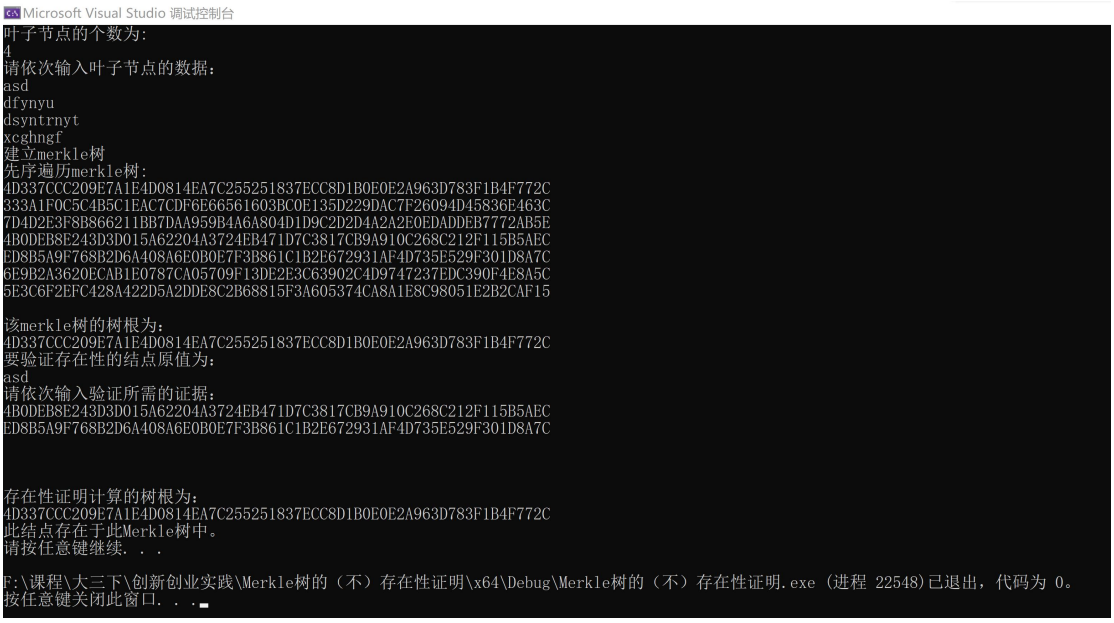
通过上述方式确实可以解决数据块奇偶的问题，并可以正确构建 Merkle 树。但是却不能处理当数据块的个数不是 2 的幂次的情况（eg:6）。即当偶数个（非 2 的幂次）的数据块进行过若干层的合并再 HASH 处理后，总会在某一层（非根部）出现 HASH 块为奇数的情况，此时不易在拿到叶子节点个数时，确定每一个叶子节点所在的数组中的索引。

因此我又对于实现思路进行了改进。引入了循环队列，通过循环队列的方式，每次步骤中总是出队列两个，然后进行合并再 HASH 后，再将这一个结果入队列，直到循环队列中只剩下一个 HASH 块（Merkle 根）为止。

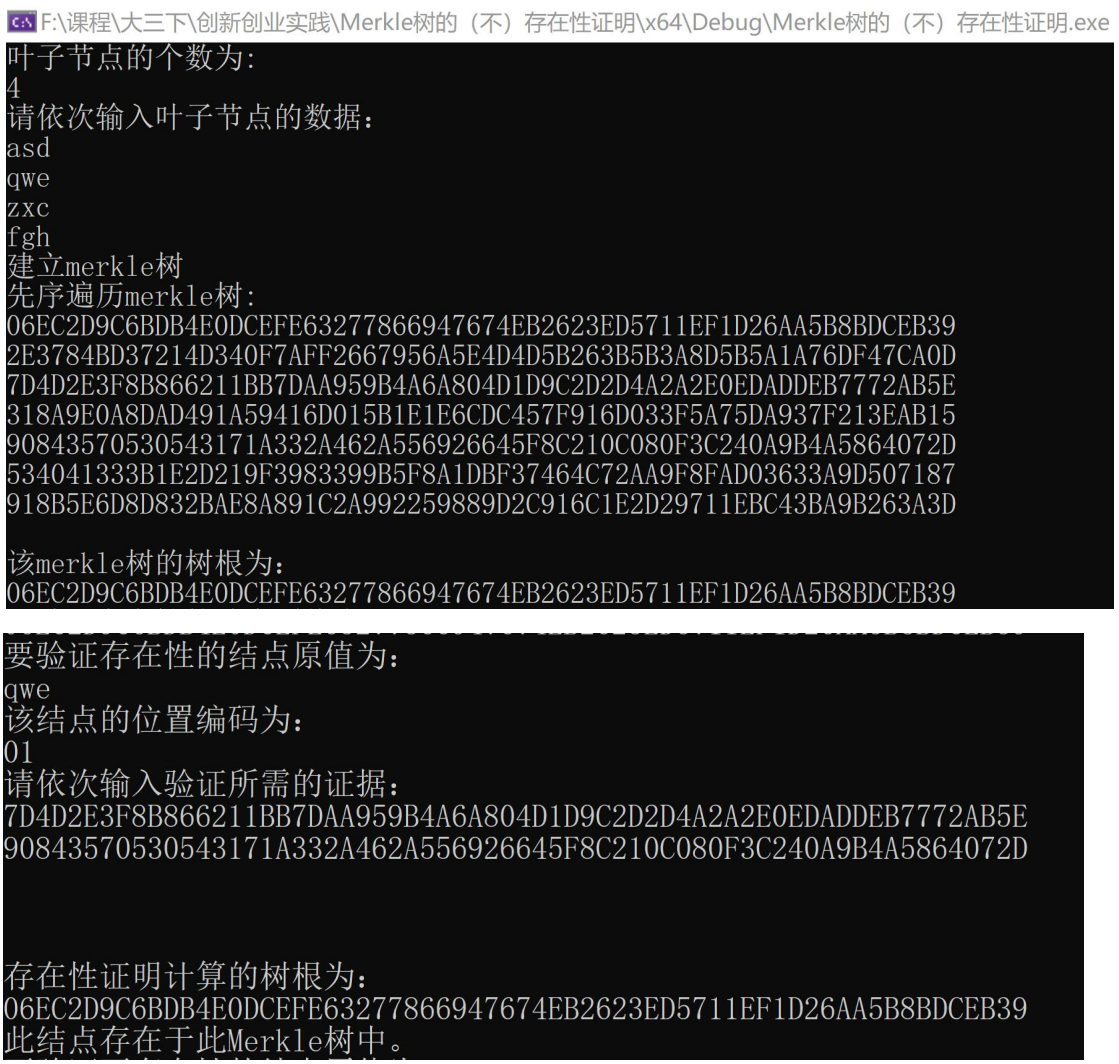
具体实现方式见“Merkle 树实现（改进）”

二、 存在性证明

在存在性证明中，对于要证明存在性的节点 x，要提供所需的证明证据（节点到 root 的路径所需的 HASH 值），根据该节点的原值和到 root 的路径所需的 HASH 值，可以通过合并再 HASH 的方式一直到计算出 Merkle 根（root）,此时，可以用计算的结果和真正的结果进行比对，从而确定此节点是否在这个 Merkel 树中。



在具体实现的过程中，我发现还需要提供要证明存在性的节点 x 的位置，从而确定在合并在计算 HASH 值时，合并的左右顺序。在本具体实现过程中，我对于位置的刻画是从根部开始，左支为 0，右支为 1，一直到节点所在的位置，在实现合并时，由于是自下而上合并，因此对于此位置要逆序依次判断操作。



三、 不存在性证明

在不存在性证明中，如果对于所有的区块进行比对，其效率是非常低下的。所以

一个想法是，找到大于该区块的最小的区块和小于该区块的最大的区块，并且在排列的 Merkle 树中两者处于相邻的位置，此时若证明这两个区块的存在性（利用上一问中的存在性证明，提供所需的响应的证据进行证明），即可证明所证区块的不存在性。

```
F:\课程\大三下\创新创业实践\Merkle树的（不）存在性证明\x64\Debug\Merkle树的（不）存在性证明.exe
叶子节点的个数为:
4
请依次输入叶子节点的数据：
asd
qwe
zxc
fgh
建立merkle树
先序遍历merkle树:
06EC2D9C6BDB4E0DCEFE63277866947674EB2623ED5711EF1D26AA5B8BDCEB39
2E3784BD37214D340F7AFF2667956A5E4D4D5B263B5B3A8D5B5A1A76DF47CA0D
7D4D2E3F8B866211BB7DAA959B4A6A804D1D9C2D2D4A2A2E0EDADDEB7772AB5E
318A9E0A8DAD491A59416D015B1E1E6CDC457F916D033F5A75DA937F213EAB15
90843570530543171A332A462A556926645F8C210C080F3C240A9B4A5864072D
534041333B1E2D219F3983399B5F8A1DBF37464C72AA9F8FAD03633A9D507187
918B5E6D8D832BAE8A891C2A992259889D2C916C1E2D29711EBC43BA9B263A3D

该merkle树的树根为：
06EC2D9C6BDB4E0DCEFE63277866947674EB2623ED5711EF1D26AA5B8BDCEB39
```

```
要验证不存在性的结点原值为：
bad
比该值小的最大的原值为：
asd
该结点的位置编码为：
00
请依次输入验证所需的证据：
318A9E0A8DAD491A59416D015B1E1E6CDC457F916D033F5A75DA937F213EAB15
90843570530543171A332A462A556926645F8C210C080F3C240A9B4A5864072D



存在性证明计算的树根为：
06EC2D9C6BDB4E0DCEFE63277866947674EB2623ED5711EF1D26AA5B8BDCEB39
比该值小的最大的值存在于此Merkle树中。
比该值大的最小的原值为：
zxc
该结点的位置编码为：
10
请依次输入验证所需的证据：
918B5E6D8D832BAE8A891C2A992259889D2C916C1E2D29711EBC43BA9B263A3D
2E3784BD37214D340F7AFF2667956A5E4D4D5B263B5B3A8D5B5A1A76DF47CA0D



存在性证明计算的树根为：
06EC2D9C6BDB4E0DCEFE63277866947674EB2623ED5711EF1D26AA5B8BDCEB39
比该值大的最小的值存在于此Merkle树中。
要证明的原值不存在于此Merkle树中。
请按任意键继续. . .
```