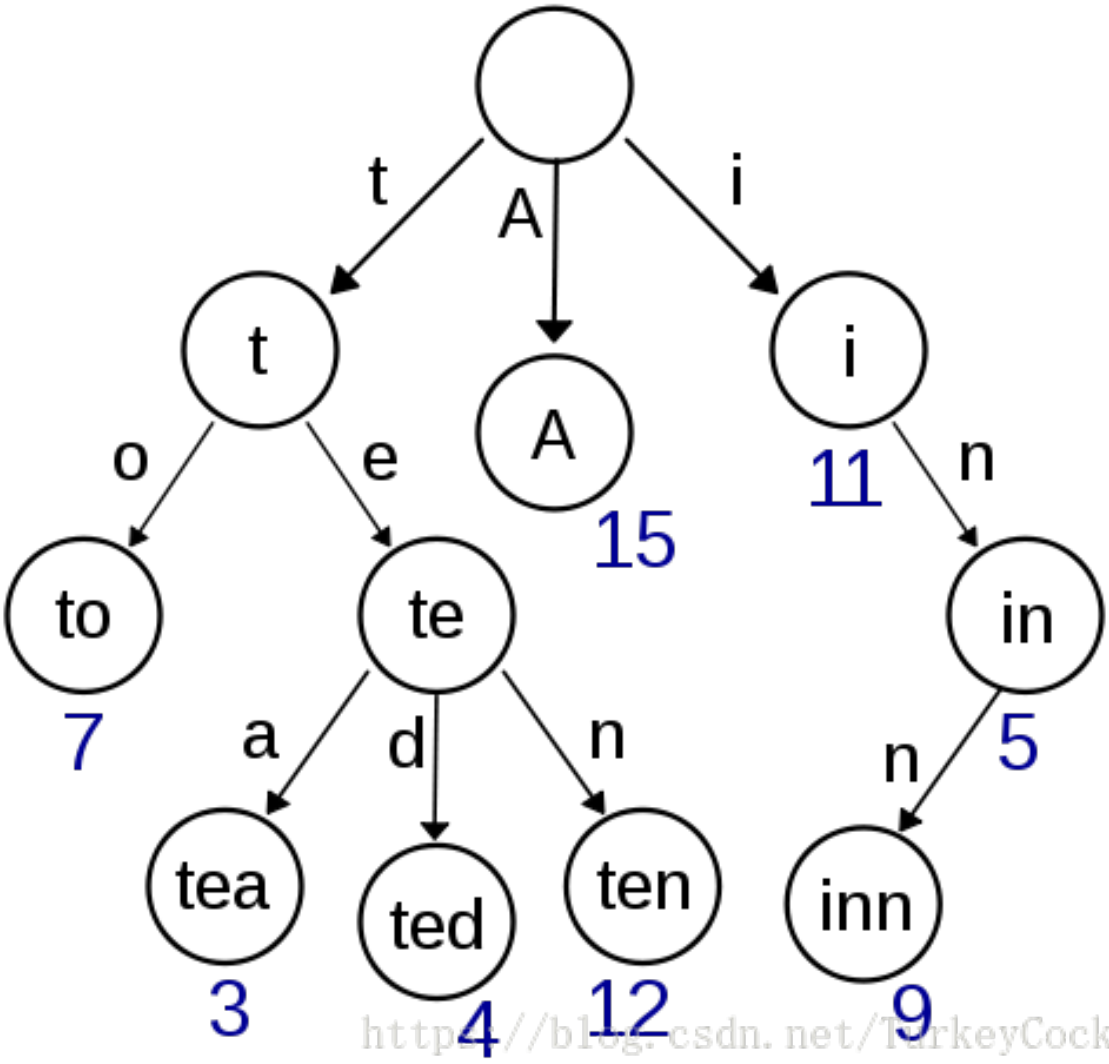
MPT全称Merkle Patricia Trie，是以太坊用来存储数据的一种数据结构。
MPT融合了Trie、Patricia Trie、Merkle Tree这3种数据结构的优点，从而实现快速查找并节省存储空间。下面依次介绍一下这几种结构的基本概念和原理。

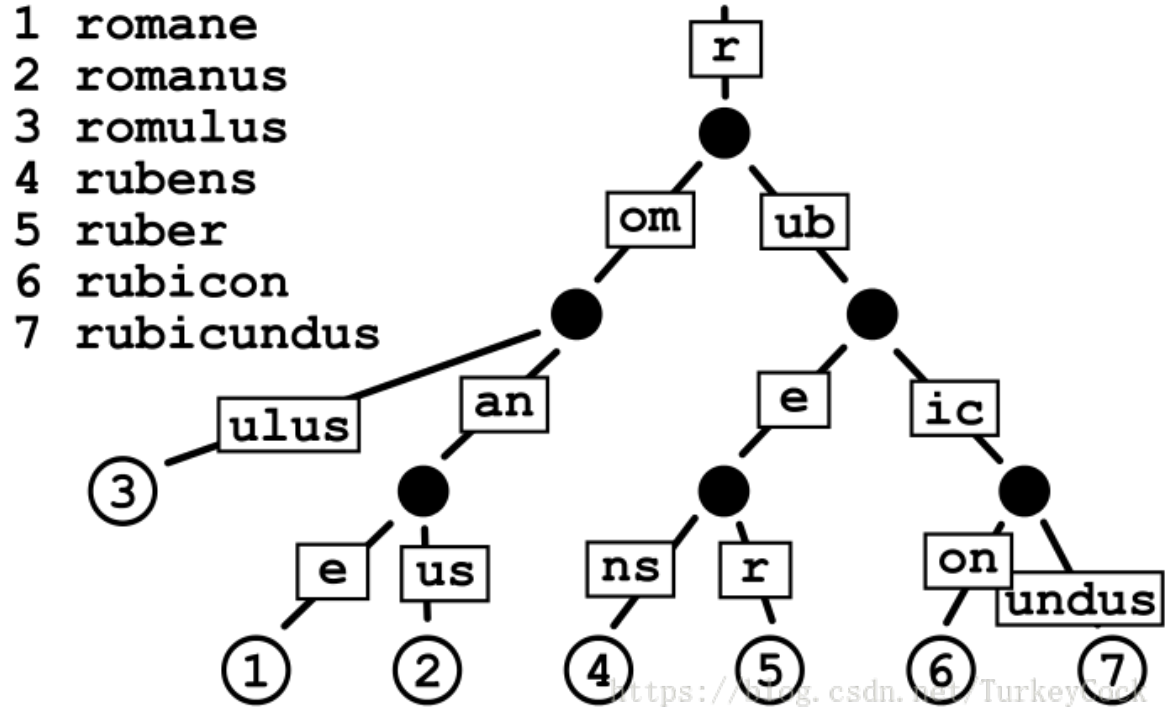
1. Trie

Trie也称为Radix Tree或者Prefix Tree，这个单词来源于retrieval(检索)这个单词的中间4个字母，中文翻译为字典树或者前缀树，是一种用于快速检索的多叉树。
原理很简单，以检索英文单词为例，只需要把每个单词按字母进行拆分然后到树上查找，查找深度和单词的长度相同。下图是一个Trie的示例：



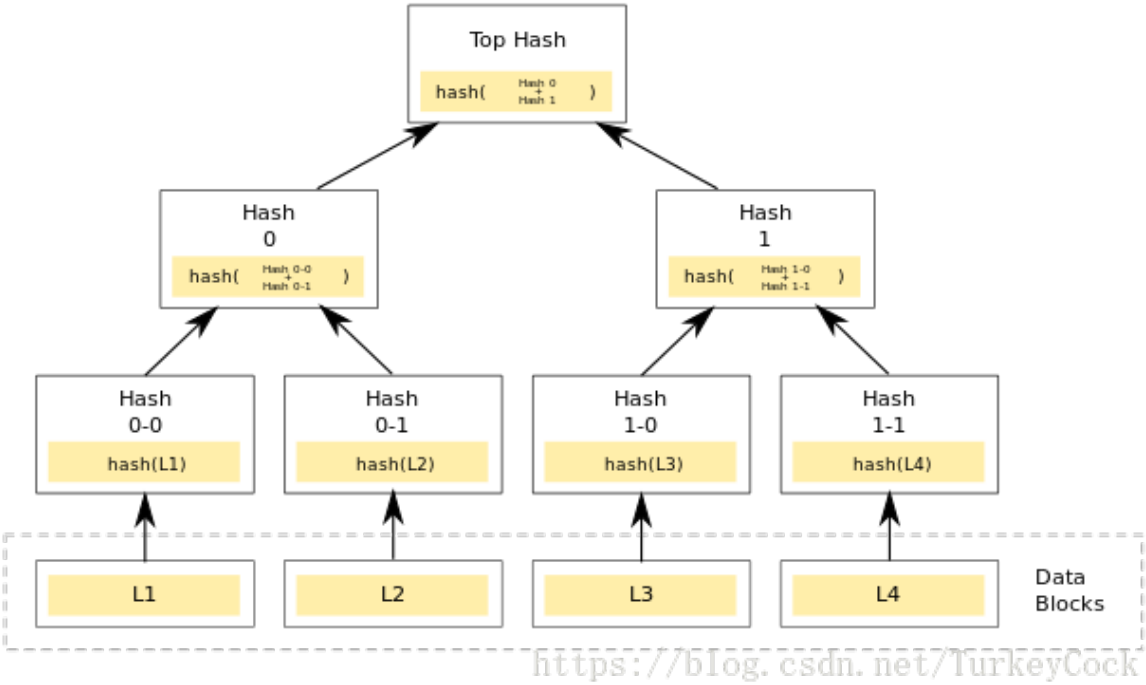
2. Patricia Trie

显然，上面这种方式需要耗费较大的存储空间，因为每个结点都必须存储26个指针指向下一级。
实际上我们并没有那么多的单词组合，有一些结点可能只有一个有效child。我们可以把这些只有一个child的结点们合并成一个结点，这就是Patricia Trie。
Patricia这个单词是一个缩写，全称是Practical Algorithm To Retrieve Information Coded In Alphanumeric。下图是一个Patricia Trie的示例：



3. Merkle Tree

Merkle Tree是一种用于快速验证内容完整性的数据结构，是有一个叫Ralph Merkle的人发明的。下图是一个Merkle Tree的示例：



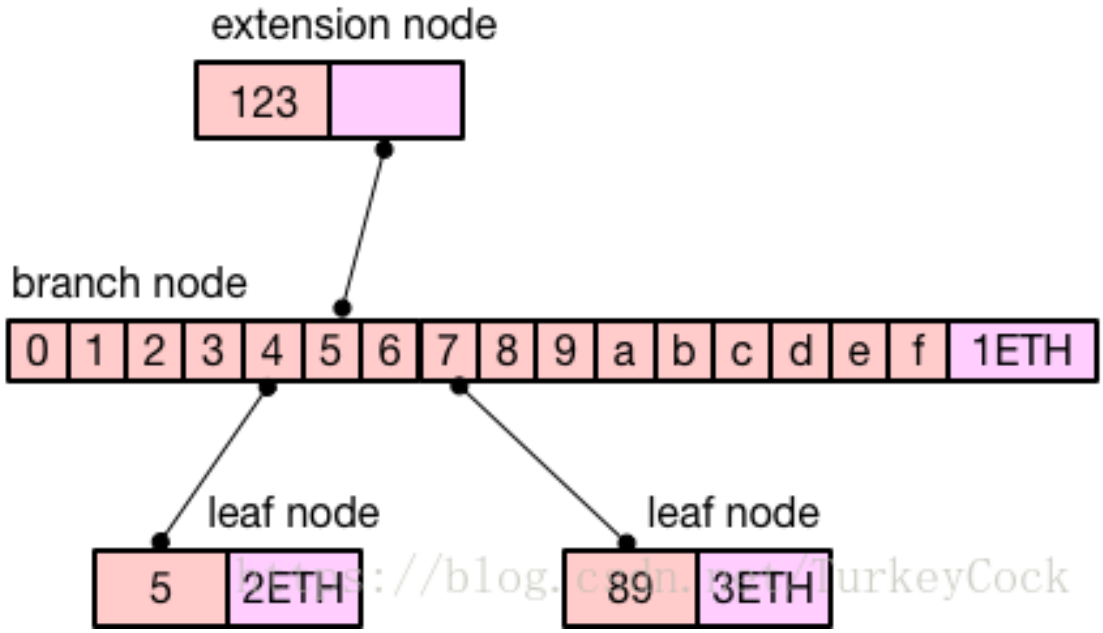
其基本原理是分别计算树的叶子结点的hash值，然后把叶子结点们的hash值拼接在一起，再计算一次hash作为其父结点的hash值，依次向上直到根结点，根结点的hash值称为Merkle Root。可以看出，如果想要获得相同的Merkle Root，所有子结点的内容都必须相同，这样就可以用来验证树的内容有没有被篡改。

4. MPT

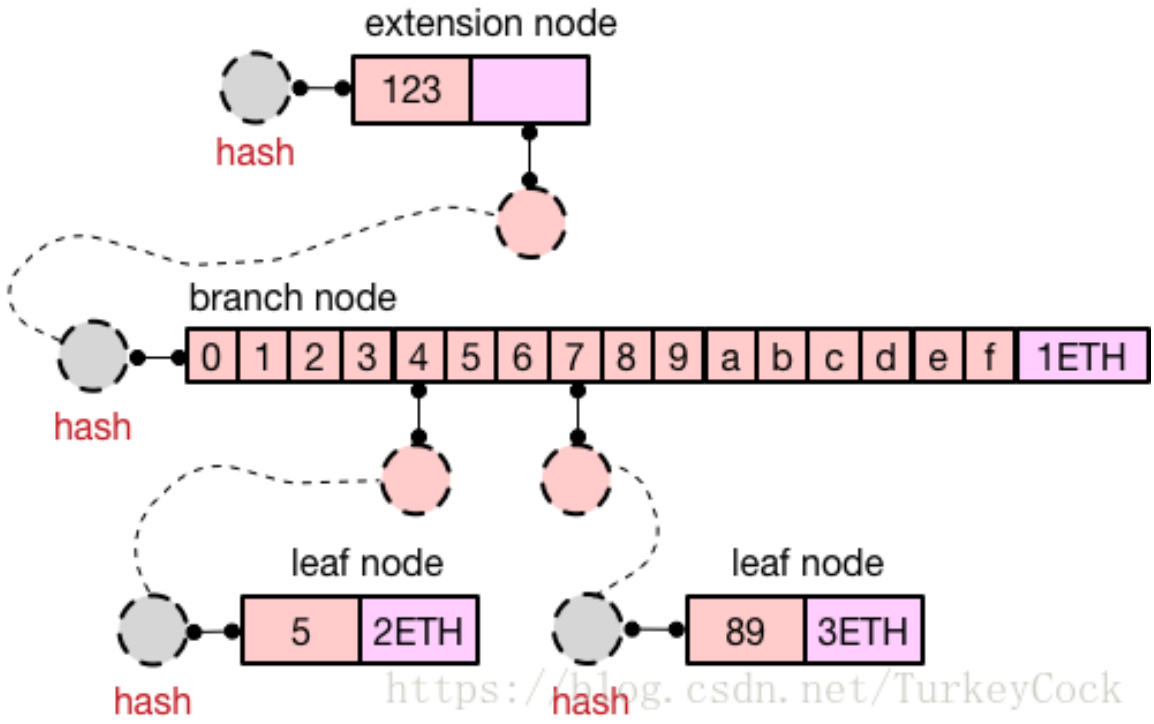
MPT融合了上面3种数据结构，但是又有一些细微的差别。
上面举的例子都是英文单词，因此是26叉树，而MPT是用来检索字节数据的，因此是16叉树，分别代表0x0~0xF。
首先MPT定义4种结点类型：

空结点（NULL）
分支结点（branch node）：包含16个分支，以及1个value
扩展结点（extension node）：只有1个子结点
叶子结点（leaf node）：没有子结点，包含一个value

举个例子：[123，’1ETH’]，[12345，’2ETH’]，[123789, ‘3ETH’]这三个数据，就会组织成下面这个样子：



最上面是一个扩展结点，体现了Patricia Trie的思想，把公共的关键字123压缩到了一个结点中。
中间的是一个分支结点，体现了Trie的思想，有16个分叉。但是由于123本身也是一个关键字，不属于任何一个分支，因此会在分支结点的最后一个字段存储对应的value。
最下面是两个叶子结点，存储对应的value。
那么Merkle Tree是怎么体现的呢？还记得吗，挖矿的时候ethash的Finalize()函数里会调用StateDB的IntermediateRoot()获取Merkle Root。这个过程中会对结点进行“折叠”（collapse），也就是把所有的子结点替换成子结点的hash值，然后再分别计算每个结点的hash值。折叠后的效果参见下图：



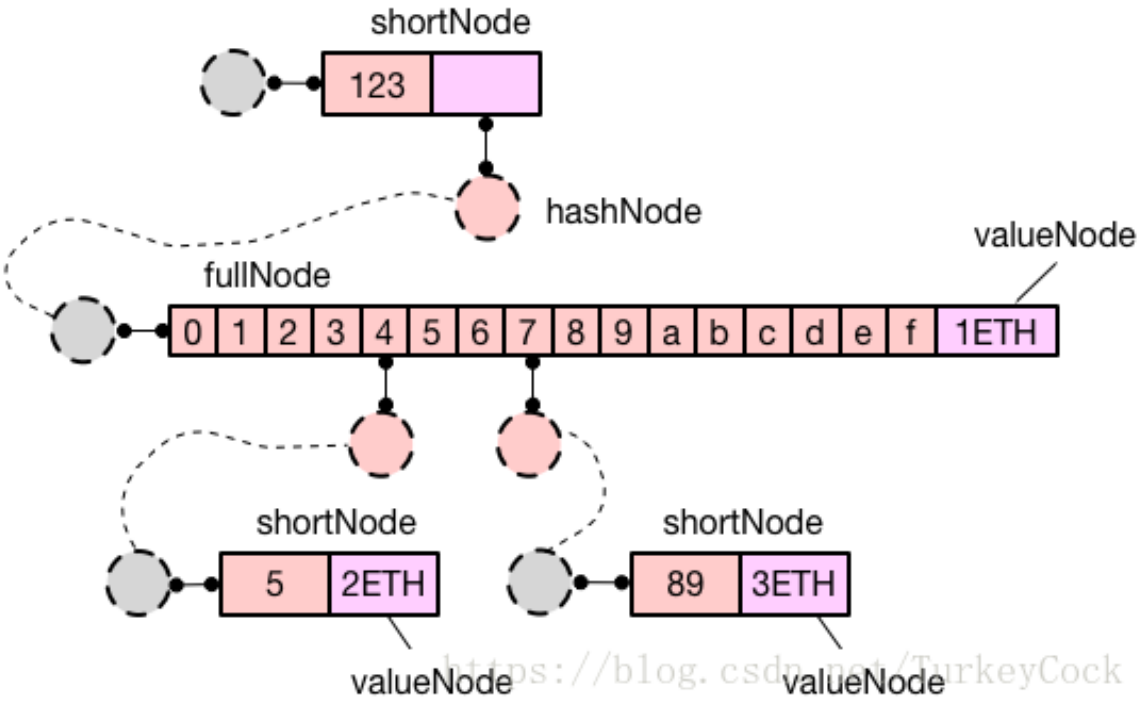
因此，我们只要有Merkle Root，就可以从数据库中恢复出根结点，再依次根据结点中存储的子结点hash值，从数据库中恢复出所有子结点。

在以太坊中，MPT是通过下面4种结点类型实现的：

```
type (
    fullNode struct {
        Children [17]node // Actual trie node data to
encode/decode (needs custom encoder)
        flags    nodeFlag
    }
    shortNode struct {
        Key   []byte
        Val   node
        flags nodeFlag
    }
    hashNode  []byte
    valueNode []byte
)
```

fullNode用于实现分支结点
shortNode用于实现扩展结点和叶子结点（区别是看Val字段指向的是valueNode还是其他类型的node）
valueNode用于存储数据（存在于fullNode或者叶子结点shortNode中）
hashNode用于实现结点的折叠

另外fullNode和shortNode中还有个nodeFlag类型的字段，用来缓存结点的hash值，或者标记结点为dirty。
因此以太坊实现的MPT结构参见下图：



在MPT执行插入操作时，可能会导致结点的分裂，比如一个shortNode裂变成一个shortNode加一个fullNode。执行删除操作时，可能导致结点的合并，比如两个shortNode合并成一个shortNode。具体逻辑参考Trie的insert()/delete()函数，这里就不

详细分析了。