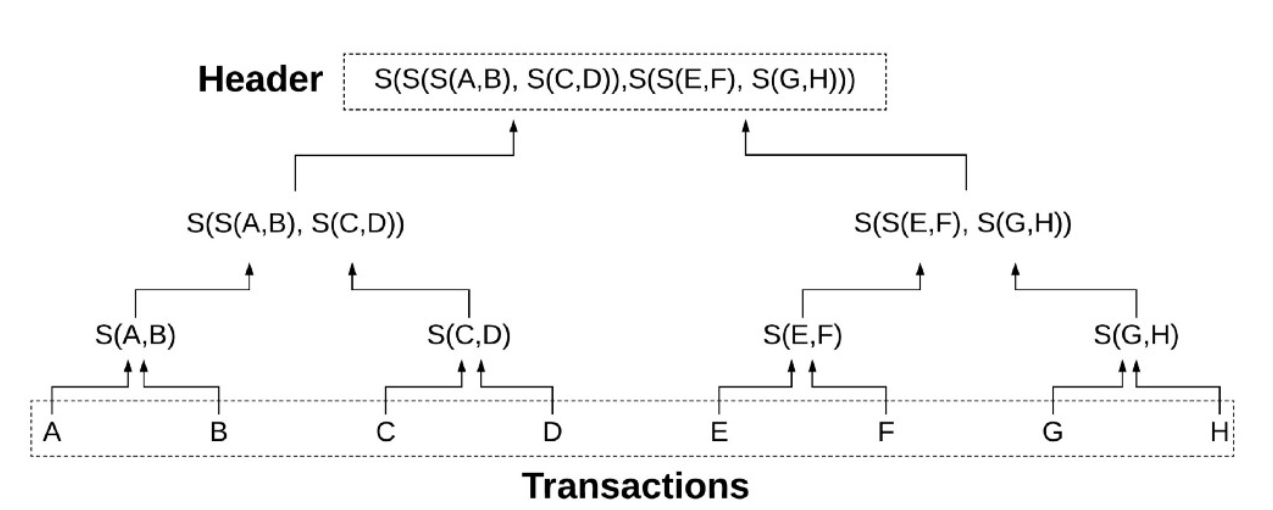
# MP研究报告

###### 基本知识背景简介：

1. 默克尔树

Merkle树是基于哈希的数据结构，用于证明存储在块中的交易数据的完整性。

该树的具体形态如下：



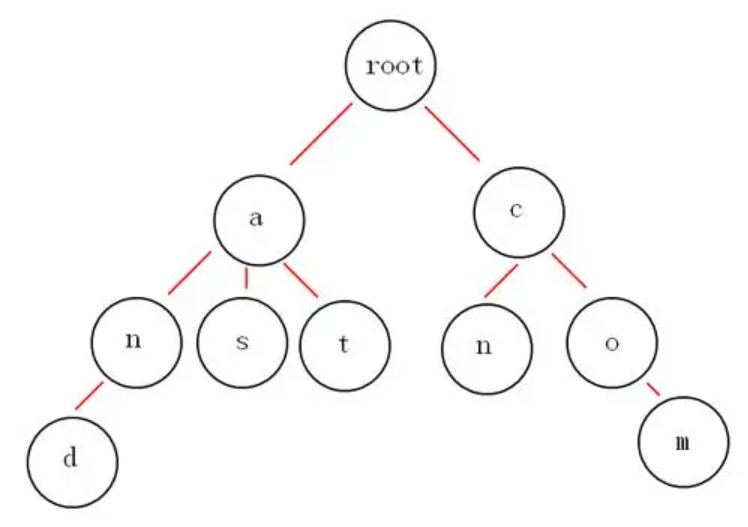
区块 （A-H） 中的八笔交易位于底层。第二行包含每两笔交易的哈希值（SHA-256、SHA-3等）；第三行包含子哈希的哈希，根包含交易哈希的哈希的哈希。一般的，这就是区块中交易部分的布局方式，这里的根即为交易头。

即该树的节点基于最下面的数据，一层一层地往上构建生成节点（left child、right child的哈希值）。如此一来，仅需提供根节点的值与正确值比较即可验证数据的正确性。【1】

2、前缀树

Trie树（前缀树）被广泛的应用在字符串匹配，搜索引擎关键词联想等问题上，

是一种多叉树，其root节点不代表字符，除了根节点以外的所有节点每个都代表一个字符，并且从根节点遍历到任意一个节点，遍历到的所有节点代表的字符连起来为该路径代表的字符串，每个节点的子节点代表的字符串各自不相同。

IMG_256

因此可以发现，前缀树把字符串存储在数的遍历路径上，而非节点上。拥有相同前缀的字符串对应的遍历前缀路径相同，这样在业务中涉及大量拥有相同前缀的字符串时可以大量复用空间，节省开销。相比于哈希表，在前缀树不会存在哈希冲突的问题。【2】

3、Patricia树

Patricia树是一种改良的数字查找树。

4、哈希树

哈希树是一种持久性数据结构，可用于实现集合和映射，旨在替换纯函数式编程中的哈希表。哈希树使用素数的模来确定数据的位置。如使用最小的10个素数，可以表示10555815270个数，这个大小已经足够包含long int类型了。对素数的余数决定了处理的路径。在其基本形式中，哈希树在trie中存储其键的哈希值（被视为位串），其中实际键和（可选）值存储在trie的“最终”节点中。哈希树的节点查找过程和节点插入过程类似，就是对关键字用素数序列取余，根据余数确定下一节点的分叉路径，直到找到目标节点。【3】

###### Merkle Patricia Tree

Merkle Patricia尝试提供一个经过加密身份验证的数据结构，可用于存储所有键值对绑定，尽管对于本文的范围，我们将键和值限制为字符串（要消除此限制，只需对其他数据类型使用任何序列化格式）。它们是完全确定性的，这意味着具有相同键值对绑定的Patricia trie保证与最后一个字节完全相同，因此具有相同的根哈希，为插入、查找和删除提供了O（log（n））复杂度的效率，并且比更复杂的基于比较的替代方案更容易理解。【4】

MPT在前缀树的基础上，对每个节点进行哈希，即每个节点是它所有子节点的哈希值（hash（child1，child2，...）），用来索引数据库和计算根节点。其索引使用了两个类型的键：前缀树的key以及默克尔树所使用的key。

MPT的主要功能如下：

·存储任意长度的key-value键值对数据

·快速计算所维护数据集哈希标识

·快速状态回滚

·默克尔证明的证明方法，进行轻节点的扩展，实现简单支付验证

对于MPT的节点类型，为使对树的查询操作和数据的增删改的效率处于高水平以及存储空间的合理节约使用，其提供了多个节点类型以压缩树高及复杂度。

MPT的节点类型如下：

·空节点

·分支节点

·叶子节点

其中当trie本身为空时，存在唯一的空节点。

分支节点用来表示MPT树中所有拥有超过1个child节点以上的非叶子节点。

根据类型不同，trie节点存储内容有稍许不同，通常会包含header、trie路径的部分key、children节点以及value。与前缀树相同，MPT同样是把key-value数据项的key编码在树的路径中，但是key的每一个字节值的范围太大，因此进行树操作之前，首先会进行一个key编码的转换，将一个字节的高低四位内容分拆成两个字节存储。通过编码转换，key的每一位的值范围都在[0, 15]内。因此，一个分支节点的孩子至多只有16个。这种方式减小了每个分支节点的容量，但是在一定程度上增加了树高。分支节点的孩子列表中，最后一个元素是用来存储自身的内容。

此外，每个分支节点会有一个附带的字段nodeFlag，记录了一些辅助数据：

·节点哈希：若该字段不为空，则当需要进行哈希计算时，可以跳过计算过程而直接使用上次计算的结果（当节点变脏时，该字段被置空）；

·脏标志：当一个节点被修改时，该标志位被置为1；

·诞生标志：当该节点第一次被载入内存中（或被修改时），会被赋予一个计数值作为诞生标志，该标志会被作为节点驱除的依据，清除内存中“太老”的未被修改的节点，防止占用的内存空间过多；

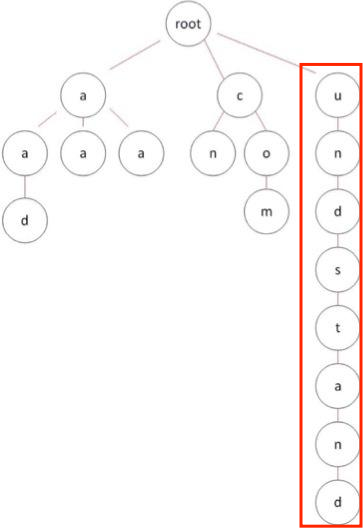
叶子节点：

其中关键的字段为：

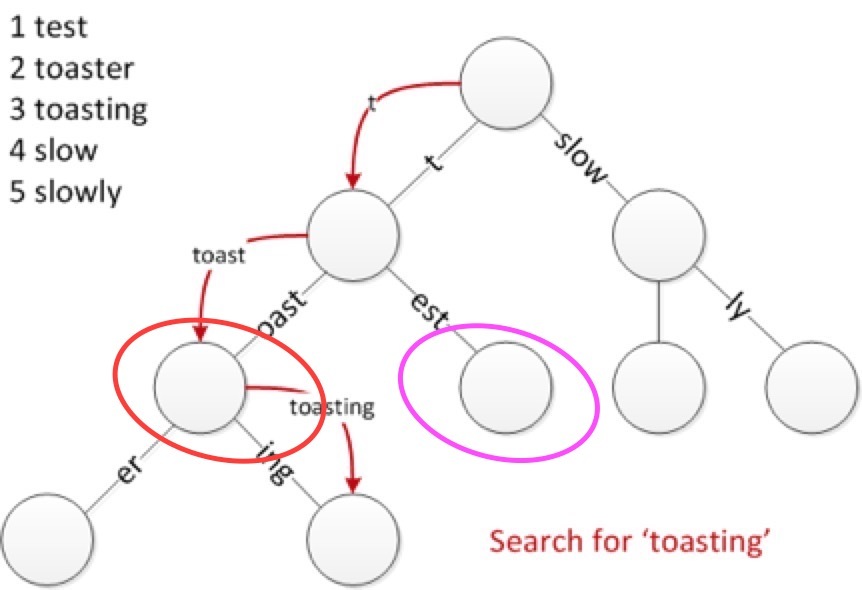
·Key：用来存储属于该节点范围的key；

·Val：用来存储该节点的内容；

使用前缀树造成的空间浪费：



右侧出现过长的子节点，它们均为非叶子节点，且仅为构建一条路径。对于此类浪费，MPT使用当插入的节点与其他的某个节点具有相同的前缀路径，则将其key储存在叶子节点的key中。



如果我们将其转化位trie结构进行表示即（Trie路径所对应的key），通用的trie构建思路为：构建trie节点，header表示该tried节点的级数，1开头表示用于索引的目录节点，0开头则表示数据节点，每个数据节点拥有自己节点对应的key

在检索数据的时候经过目录trie节点的时候需要将该节点对应的部分key值（这里使用了两位，作为key的前缀）加入带检索的key值，经过数据节点的时候，则将其key值作为当前检索的key值的剩余位级联到后面。

【5】

###### 具体操作

以如下MPT为例：

其中从根节点开始搜寻与搜索路径内容一致的路径：

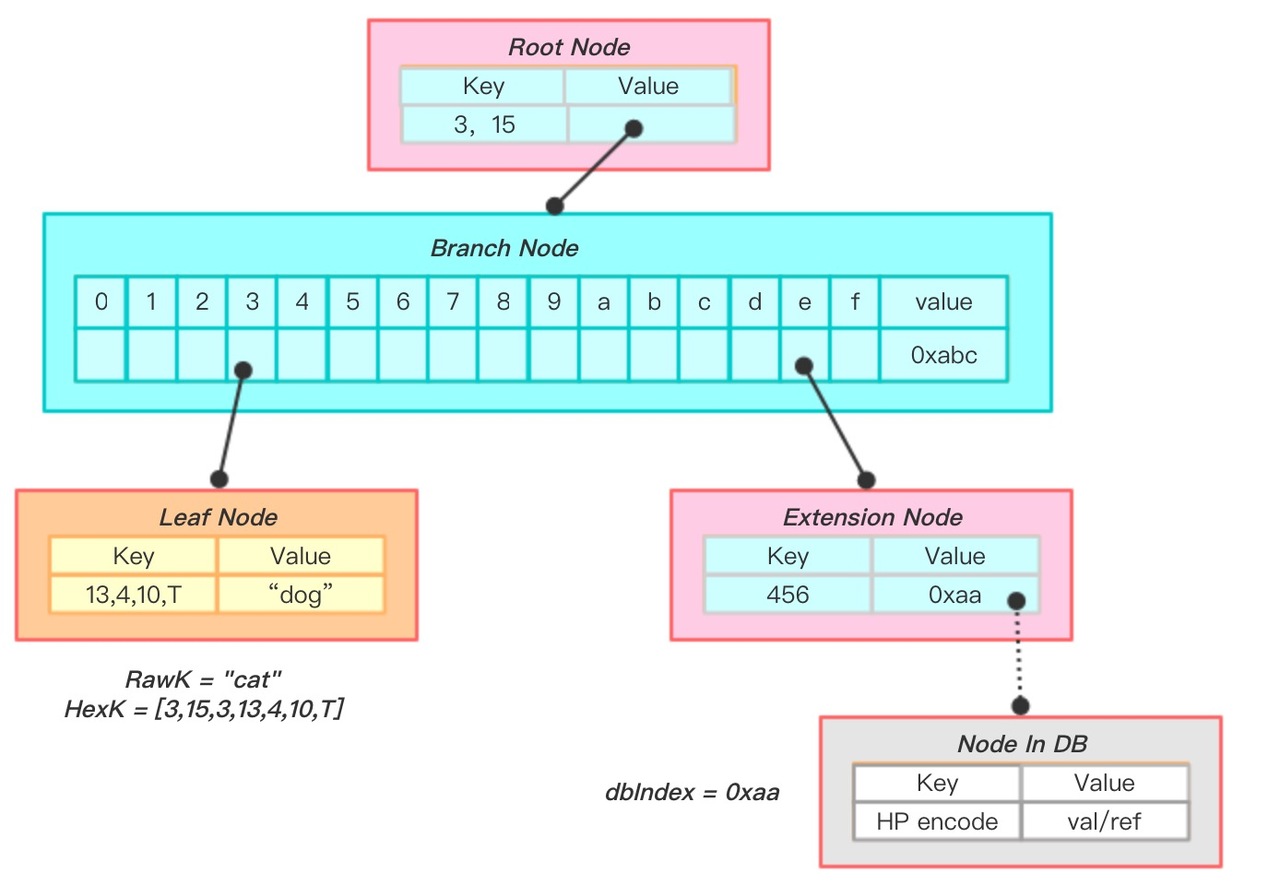
·如果当前节点是叶节点，则存储的内容是数据元素的内容，并且搜索路径的内容与叶节点的关键字相同，然后查找节点。相反，它表示树中不存在节点。

·如果当前节点是扩展节点，存储的内容是哈希索引，则哈希索引用于从数据库加载节点，然后搜索路径用作参数，递归调用搜索函数以分析新节点。

·如果当前节点是扩展节点，则存储的内容是对另一个节点的引用，当前节点的键是搜索路径的前缀，然后通过从当前节点中减去键来搜索路径，其余的以搜索路径作为参数，搜索函数通常调用子节点。如果当前节点的键不是搜索路径的前缀，则表示该节点不在树中。

·如果当前节点是分支节点，如果搜索路径为空，则返回分支节点的存储内容。相反，搜索路径的第一个字节用于识别分支节点的子节点，其余的搜索路径用作参数，递归地调用搜索函数。

·接着，我们对每个节点计算出对应的哈希，首先计算出叶子节点的哈希值，它们被上一级的分支节点所引用，用来在数据库中查找对应的节点；然后计算分支节点的哈希值，直至递归抵达根节点。



·将cat转换成hex编码[3,15,3,13,4,10,T] （在末尾添加终止符是因为需要查找一个真实的数据项内容）；

·当前节点是根节点，且是扩展节点，其key为3,15，则递归地对其子节点进行查找调用，剩余的搜索路径为[3,13,4,10,T];

·当前节点是分支节点，以搜索路径的第一个字节内容3选择第4个孩子节点递归进行查找，剩余的搜索路径为[13,4,10,T]；

·当前节点是叶子节点，且key与剩余的搜索路径一致，表示找到了该节点，返回Val为dog。

插入操作也是基于查找过程完成的，一个插入过程为：

·首先找到与新插入节点拥有最长相同路径前缀的节点，记为Node；

若该Node为分支节点：

·剩余的搜索路径不为空，则将新节点作为一个叶子节点插入到对应的孩子列表中；

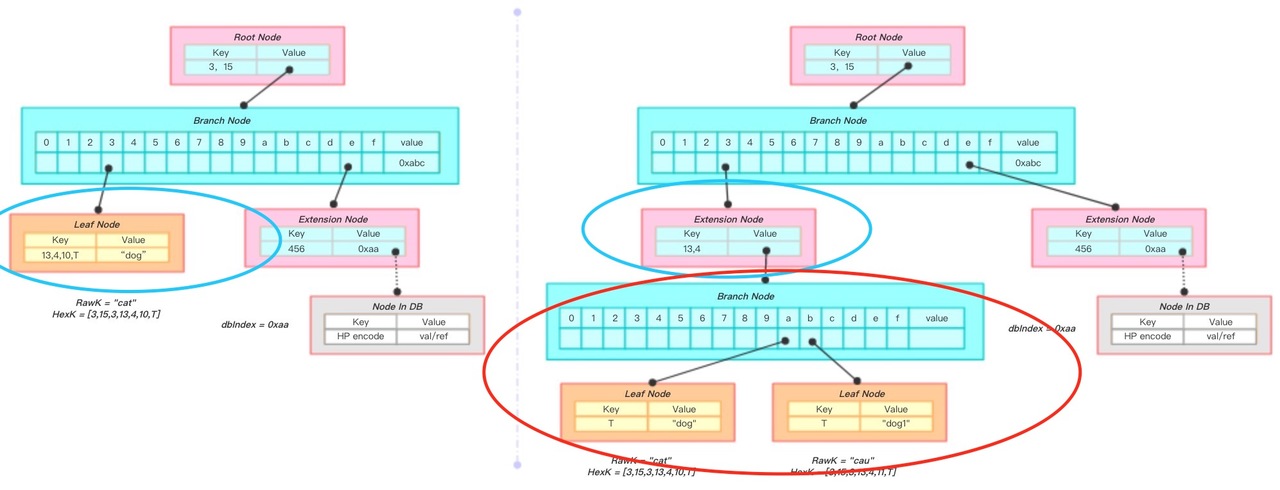
·剩余的搜索路径为空（完全匹配），则将新节点的内容存储在分支节点的第17个孩子节点项中（Value）；

若该节点为叶子／扩展节点：

·剩余的搜索路径与当前节点的key一致，则把当前节点Val更新即可；

·剩余的搜索路径与当前节点的key不完全一致，则将叶子／扩展节点的孩子节点替换成分支节点，将新节点与当前节点key的共同前缀作为当前节点的key，将新节点与当前节点的孩子节点作为两个孩子插入到分支节点的孩子列表中，同时当前节点转换成了一个扩展节点（若新节点与当前节点没有共同前缀，则直接用生成的分支节点替换当前节点）；

·若插入成功，则将被修改节点的dirty标志置为true，hash标志置空（之前的结果已经不可能用），且将节点的诞生标记更新为现在；



上图是一次将key为“cau”, value为“dog1”节点插入的过程。

·将key"cau"转换成hex编码[3,15,3,13,4,11,T] ；

·通过查找算法，找到左图蓝线圈出的节点node1，且拥有与新插入节点最长的共同前缀[3,15,3,13,4]；

·新增一个分支节点node2，将node1的val与新节点作为孩子插入到node2的孩子列表中，将node1的val替换成node2；

·node1变成了一个扩展节点；

删除操作与插入操作类似，都需要借助查找过程完成，一次删除过程为：

根据插入操作描述的查找步骤，找到与需要插入的节点拥有最长相同路径前缀的节点，记为Node；

若Node为叶子／扩展节点：

·若剩余的搜索路径与node的Key完全一致，则将整个node删除；

·若剩余的搜索路径与node的key不匹配，则表示需要删除的节点不存于树中，删除失败；

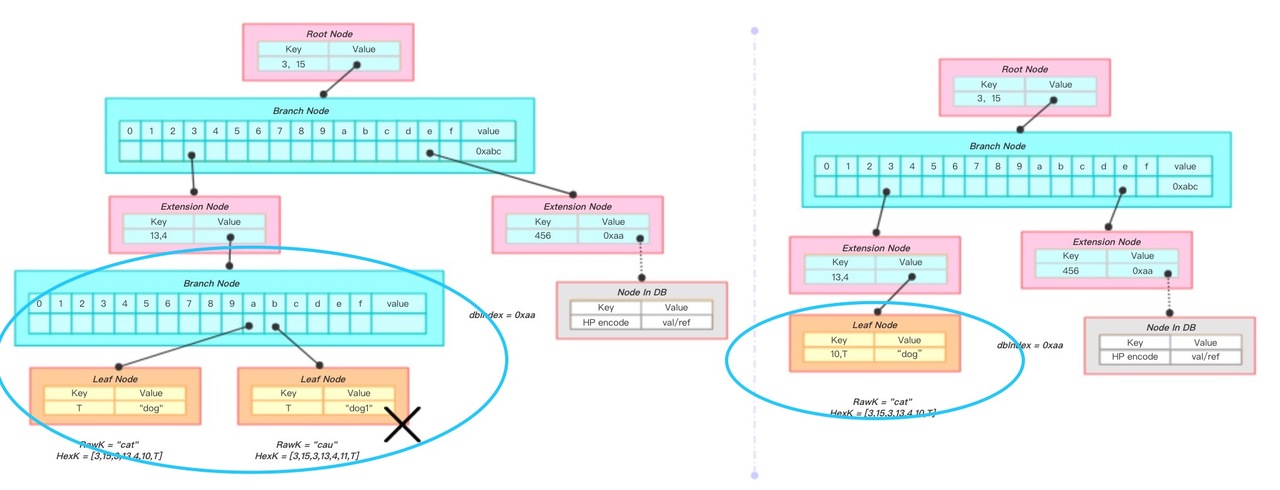
·若node的key是剩余搜索路径的前缀，则对该节点的Val做递归的删除调用；

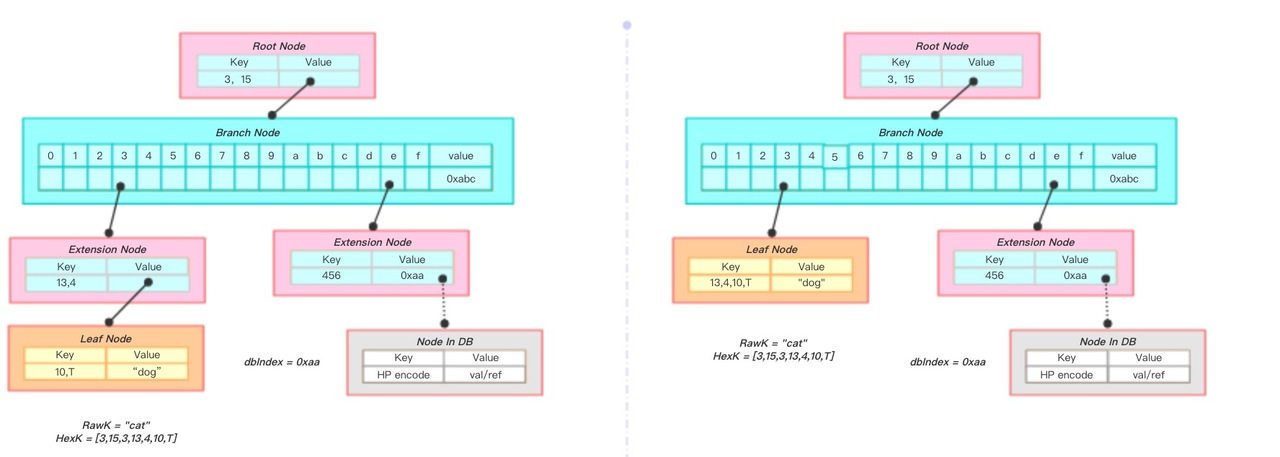
若Node为分支节点：

·删除孩子列表中相应下标标志的节点；

·删除结束，若Node的孩子个数只剩下一个，那么将分支节点替换成一个叶子／扩展节点；

·若删除成功，则将被修改节点的dirty标志置为true，hash标志置空（之前的结果已经不可能用），且将节点的诞生标记更新为现在；





上面两幅图是一次将key为“cau”, value为“dog1”节点删除的过程。

将key"cau"转换成hex编码[3,15,3,13,4,11,T] ；

通过查找算法，找到用叉表示的节点node1，从根节点到node1的路径与搜索路径完全一致；

从node1的父节点中删除该节点，父节点仅剩一个孩子节点，故将父节点转换成一个叶子节点；

新生成的叶子节点又与其父节点（扩展节点）发生了合并，最终生成了一个叶子节点包含了所有的信息（图2）；

更新操作就是Insert与Delete的结合。当用户调用Update函数时，若value不为空，则隐式地转为调用Insert；若value为空，则隐式地转为调用Delete。【5】

###### 研究总结

Merkle Patricia Trie是区块链的底层结构，世界状态、交易、交易收据等都是以这种树的形式存储在区块链数据库中，并将树root hash保存在区块头里。

【6】

###### 参考资料

【1】[Blockchain-for-Developers/merkle-tree： Merkle Tree Implementation for Lab07 (github.com)](https://github.com/Blockchain-for-Developers/merkle-tree)

【2】[前缀树(字典树）\_小轩爱学习的博客-CSDN博客\_前缀树和字典树](https://blog.csdn.net/weixin_45746505/article/details/123542664)

【3】[哈希树\_百度百科 (baidu.com)](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E6%A0%91/8832600?fr=aladdin)

【4】[Patricia Tree\_跨链技术践行者的博客-CSDN博客\_patricia tree](https://blockchain-fans.blog.csdn.net/article/details/90300577?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~default-1-90300577-blog-121735619.pc_relevant_multi_platform_featuressortv2dupreplace&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~default-1-90300577-blog-121735619.pc_relevant_multi_platform_featuressortv2dupreplace&utm_relevant_index=1)

【5】[Merkle Patricia Tree详解\_tianlongtc的博客-CSDN博客\_patricia tree](https://blog.csdn.net/tianlongtc/article/details/80418923)

<https://github.com/ethereum/wiki/wiki/Patricia-Tree>

<http://gavwood.com/paper.pdf>

<https://zh.wikipedia.org/wiki/Trie>

<https://en.wikipedia.org/wiki/Merkle_tree>

<https://github.com/ethereum/wiki/wiki/Design-Rationale>

<https://github.com/ethereum/wiki/wiki/Light-client-protocol>

【6】[1\_merkle-tree-master.rar-网络攻防文档类资源-CSDN文库](https://download.csdn.net/download/guoruijun_2012_4/12831396?spm=1001.2101.3001.5697" \t "https://blog.csdn.net/weixin_35551323/article/details/_blank)