*Project22: research report on MPT

介绍

Merkle Patricia Tree (MPT)，也被称为 Merkle Patricia Trie，是一种结合了前缀树（Trie）和默克尔树（Merkle Tree）的数据结构，它在以太坊中被用来组织和管理账户数据以及生成交易集的重要数据结构哈希。以下是对 MPT 的功能进行解释：

1.存储任意长度的 key-value 键值数据：MPT 允许存储任意长度的数据，其中键值对是通过 key 进行关联的。这些键值对可以是账户的状态数据、交易信息等。

2．提供快速计算维护数据集哈希的机制：MPT 的一个关键特点是可以高效地计算数据集的哈希值。这使得在以太坊这样的去中心化区块链系统中，可以快速验证整个数据集的完整性。

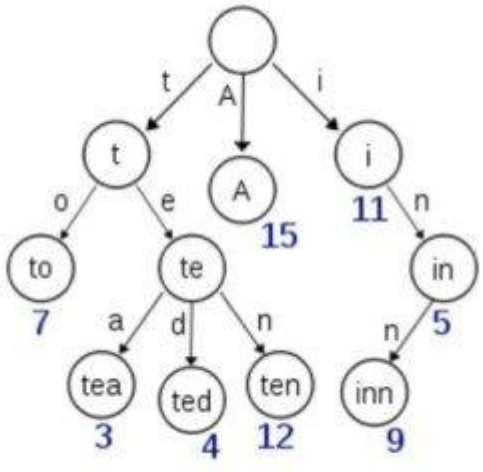
3.提供快速状态的滚动机制：MPT 允许对数据集进行状态的更改和滚动更新。在区块链中，当有新的交易发生时，只需要更新部分数据而不是整个数据集，这样可以显著提高处理效率。

4.提供一种叫默克尔证明的方法，轻量级扩展节点，简单的支付验证：MPT 基于默克尔证明（Merkle Proof）的概念，允许在不访问所有数据的情况下验证特定数据是否在树中。这使得轻客户端可以快速验证交易和账户状态，而无需完整地同步整个区块链。

MPT 结合了前缀树和 Merkel Tree 两种树结构的特点和优点，下面先来了解这两种树。

前缀树

前缀树是一种树形数据结构，特别适合存储字符串集合，其中共享相同前缀的字符串会共享相同的前缀树节点，从而节省存储空间。在 MPT 中，前缀树的特点用于存储键值数据，以便快速查找和定位特定的账户或



交易。

前缀树在某些方面具有优势，但也有一些明显的缺点。

优点：

高效查询公共前缀键数据：对于具有公共前缀的键值数据，前缀树能够高效地进行查询，因为它可以共享相同前缀的节点，减少了搜索的范围。

避免哈希冲突：前缀树不会像哈希表一样出现哈希冲突问题，因为数据的存储是基于键的前缀而不是哈希值。

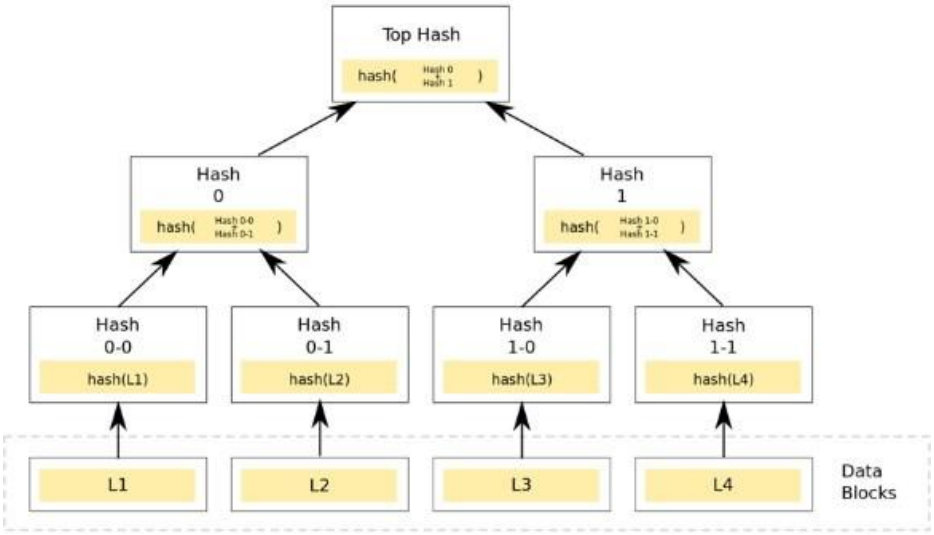
缺点：

直接查找效率较低：在最坏的情况下，当要查找的前缀为空字符串或与树中大多数键值数据没有公共前缀时，前缀树的效率会下降，需要遍历整棵树。相比之下，哈希表的查找效率是常数时间 O(1)。

可能造成空间浪费：当有一个节点的键值很长，而且与树中的其他节点没有公共前缀时，前缀树需要创建许多非叶子节点来构建路径，可能会造成存储空间的浪费。

默克尔树

默克尔树是一种用于验证数据完整性的二叉树结构，其中每个非叶节点的值是其子节点值的哈希。默克尔树在区块链中被广泛使用，特别是在比特币和以太坊等区块链中，用于验证区块数据的完整性和有效性。在 MPT 中，默克尔树的概念用于生成整个数据集的根哈希，从而便于快速



验证数据的完整性。

在比特币网络中，默克尔树就是数据项的值，而数据项的值就是数据项的哈希值。自下而上构建。下面的例子中，先对 L1-L4 的四个单元数据进行哈希处理，然后将哈希值存储到对应的叶子节点中。这些节点分别是 Hash0-0、Hash0-1、Hash1-0、Hash1-1 将两个相邻节点的 hash 合并成一个字符串，然后计算该字符串的 hash，这两个节点就是父节点的 hash 值

如果两棵树的根哈希一致，那么两棵树的结构、节点的内容一定是相同的。如上图所示，一棵树有四个叶子节点，计算代表整棵树的 hash 需要经过 7 次计算。

同样，默克尔树也存在鲜明的特点：

快速重新哈希：默克尔树的结构使得在树中节点内容发生改变时，只需要对修改后的节点重新计算哈希值，就能快速得到新的根哈希，用来表示整棵树的状态。这样的特性在区块链中非常有用，因为区块链数据是不可变的，当有新的交易添加到一个区块时，只需对受影响的节点重新计算哈希值，而无需重新计算整个区块的哈希。

轻节点扩展：默克尔树使得在公链环境下扩展轻节点成为可能。轻节点只需要存储区块头数据，因此占用存储空间较小。通过轻节点，可以在不可信的公链环境下验证一笔交易是否包含在区块链账本中，而不需要下

载完整的区块数据。这为在资源受限的终端设备上运行区块链客户端提供了解决方案。

存储空间开销大：默克尔树的一个主要缺点是它需要额外的存储空间来存储节点的哈希值。每个节点的哈希值都会占用一定的空间，因此在构建大型的默克尔树时，存储空间开销会相对较大。不过在实际应用中，这个缺点往往可以被接受，因为默克尔树在区块链和其他一些领域中的优势往往能够弥补这个缺点。

节点分类

前缀树虽然可以起到维护键值数据的目的，但是它有一个非常明显的局限性。无论是查询操作，还是增减数据，不仅效率低下，而且存储空间占用严重。因此，在以太坊中，为 MPT 树添加了几种不同类型的树节点，以尽量压缩整体树的高度，降低操作的复杂度。

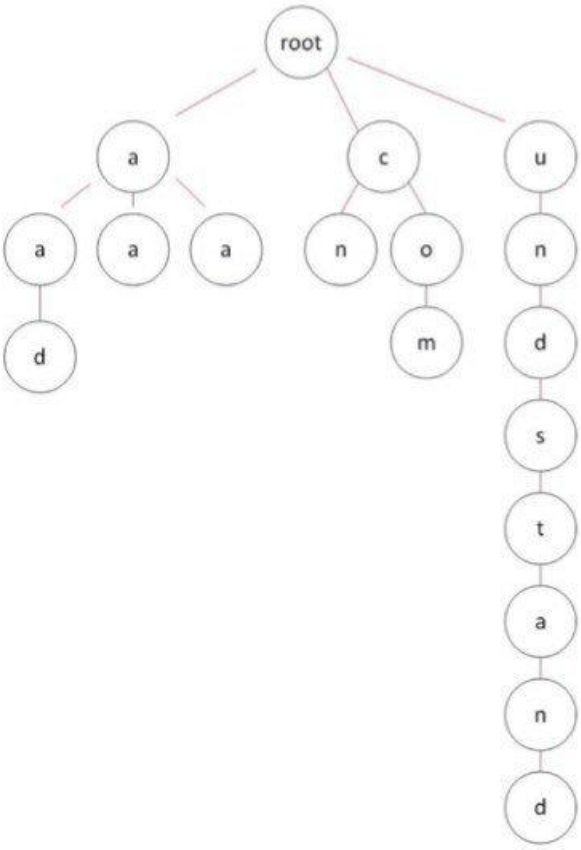
在 MPT 树中，树节点可以分为以下四类：

(1)空节点：其空心节点用于表示空字符串。

(2)分支节点：分支节点用于表示 MPT 树中具有多个子节点的所有非叶节点。分支节点的子列表中的最后一个元素用于存储其自身的内容。

(3)叶子节点&扩展节点

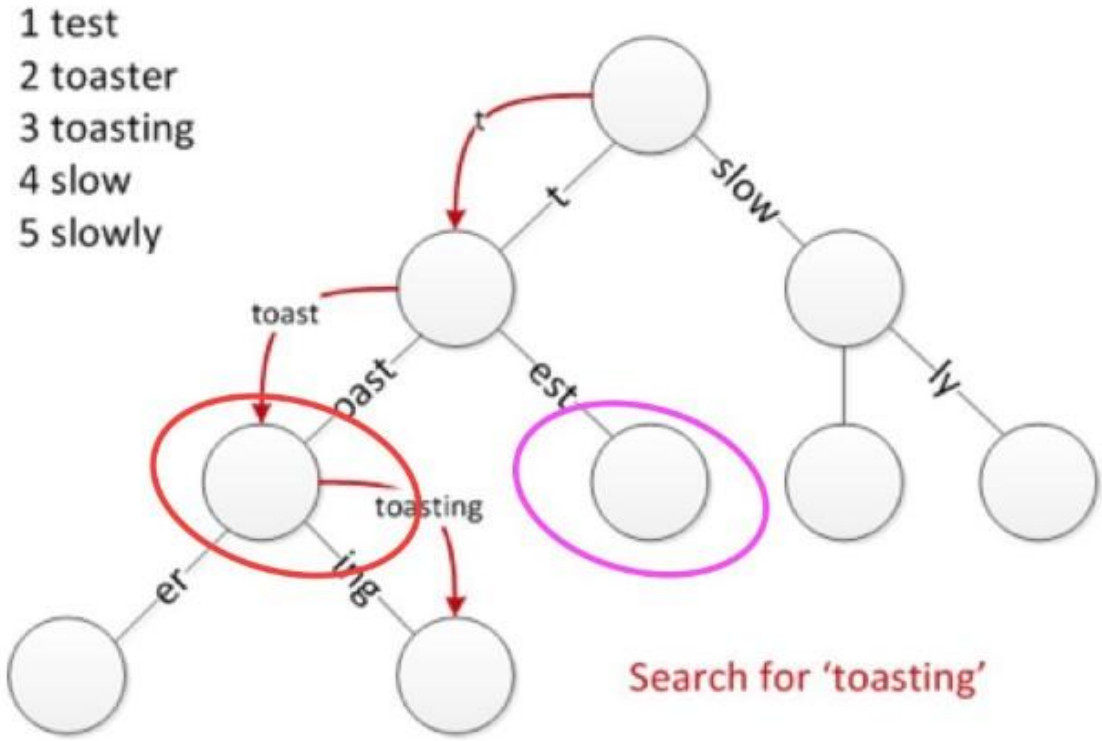
叶子节点与扩展节点的定义类似。前面提到，前缀树会存在存储空间的严重浪费，如下图：



图中右侧有一长串节点，其中大部分只是非叶子节点，用于构建一条路径，只存储该节点路径上的叶子。

在这种情况下，MPT 树对此进行了优化：当 MPT 尝试插入节点时，插入过程发现当前不存在与节点 Key 具有相同前缀的路径。此时，MPT 将剩余的密钥存储在叶/扩展节点的 Key 字段中，并充当“快捷方式”。

例如我们将红线的节点称为 node1，蓝线的节点称为 node2。node1 和 node2 共享路径前缀 t，但是 node1 在插入时，树中并没有与 oast 路径共同的前缀，因此 node1 以 oast 为键，实现编码路径压缩。



这种方法有以下优点：

(1)提高节点查找效率，避免过多的磁盘访问；

(2)减少存储空间浪费，避免存储无用节点；

键值编码

在以太坊中，MPT 树键值共有三种不同的编码方式，以满足不同场景的不同需求。

三种编码方式是：

(1) 原始编码（原生字符）：Raw Encoding 是原始的键值，不要做任何改变。这种按键编码方式，MPT 对外接口提供默认的编码方式。

例如，键为“cat”，值为“dog”数据项，Raw 代码为 ['c', 'a', 't']，替换为 ASCII 表示为 [63, 61, 74]

（2）十六进制编码（scalablecoding hex）：在分支节点中，为了减少子节点的分支数量，需要对键码进行转换，将原来键的高低四位拆分成两个字节进行存储。这种转换的关键编码，即 Hex 编码。

（3） Hex-encoding the Prefix（十六进制前缀编码）：节点加载到内存中时，还需要通过额外的机制来区分节点的类型。因此 etherbox 提出了一种 HP 编码来区分数据库中存储的叶子/扩展节点的密钥。这两类节点持久化到数据库之前，都会将节点的 key 进行编码转换，即从 Hex 编码转换为 HP 编码。

MPT 安全性

MPT 树可用于存储任意长度的键值数据项的内容。如果数据项 key 的长度不限制，当树中维护的数据量很大时，仍然会导致整棵树的深度变得越来越深，会造成以下影响：

（1）查询一个节点可能需要很多读 IO 次数，效率低下；

（2）Dos 系统容易受到攻击，攻击者通过“构造”一棵很长路径的树来存储合约中的具体数据，然后调用 SLOAD 指令不断读取树中节点的内容，导致系统效率下降极度减少；

（3）所有的密钥实际上都是明文的存储形式；

为了解决这个问题，在 MPT 以太坊的平方中再封装一个数据项（sha3（key），value）。使得传入 MPT 接口的密钥是可以避免的固定长度（32 字节），以便将密钥传递到 MPT 接口。但是，缺点是每棵树需要操作一次，以增加哈希计算量；同时，需要在数据库中存储额外的 SHA3(密钥)与密钥的对应关系。