Merkle Patricia Tree 的基本概述

Merkle Patricia Tree（又称为 Merkle Patricia Trie）是一种经过改良的、融合了默克尔树和前缀树两种树结构优点的数据结构，是以太坊中用来组织管理账户数据、生成交易集合哈希的重要数据结构。

MPT 树有以下几个作用：

存储任意长度的 key-value 键值对数据；

提供了一种快速计算所维护数据集哈希标识的机制；

提供了快速状态回滚的机制；

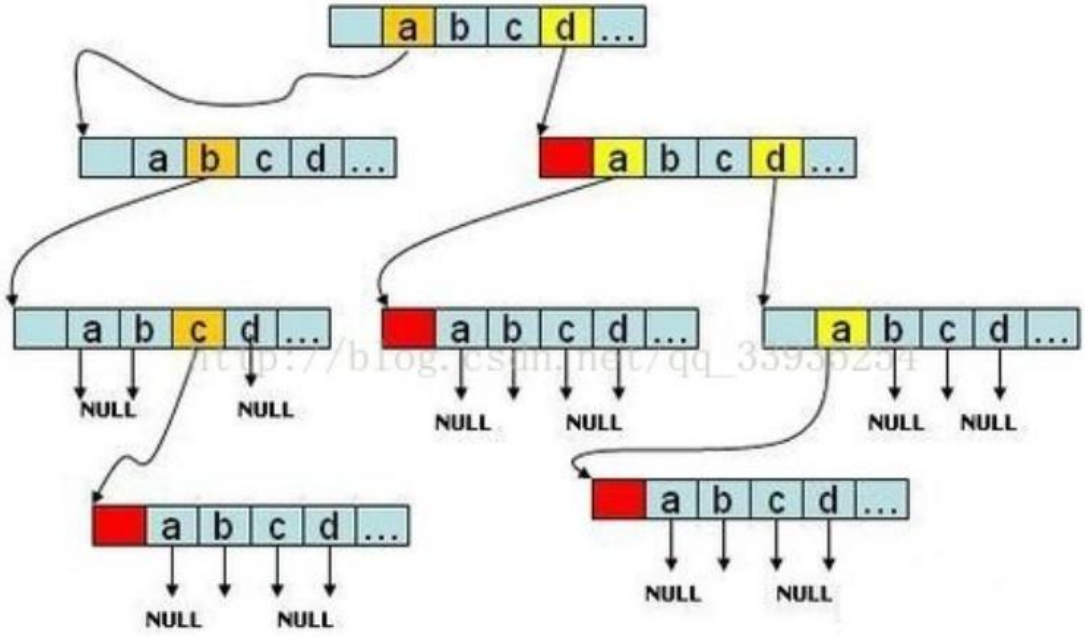
提供了一种称为默克尔证明的证明方法，进行轻节点的扩展，实现简单支付验证；

由于 MPT 结合了（1）前缀树（2）默克尔树两种树结构的特点与优势 ，因此在介绍 MPT 之前，我们首先简要地介绍下这两种树结构的特点。

前缀树的基本介绍

前缀树（又称字典树），用于保存关联数组，其键（key）的内容通常为字符串。前缀树节点在树中的位置是由其键的内容所决定的，即前缀树的 key 值被编码在根节点到该节点的路径中。

常见的用来存英文单词的 trie 每个节点是一个 **长度为 27 的指针数组，index0-25 代表 a-z 字符，26 为标志域** 。如图：



(1)优势：

相比于哈希表，使用前缀树来进行查询拥有共同前缀 key 的数据时十分高效，例如在字典中查找前缀为 pre 的单词，对于哈希表来说，需要遍历整个表，时间效率为 O(n)；然而对于前缀树来说，只需要在树中找到前缀为 pre 的节点，且遍历以这个节点为根节点的子树即可。

但是对于最差的情况（前缀为空串），时间效率为 O(n)，仍然需要遍历整棵树，此时效率与哈希表相同。

相比于哈希表，在前缀树不会存在哈希冲突的问题。

(2)劣势：

直接查找效率低下

前缀树的查找效率是 O(m)，m 为所查找节点的 key 长度，而哈希表的查找效率为 O(1)。且一次查找会有 m 次 IO 开销，相比于直接查找，无论是速率、还是对磁盘的压力都比较大。

可能会造成空间浪费

当存在一个节点，其 key 值内容很长（如一串很长的字符串），当树中没有与他相同前缀的分支时，为了存储该节点，需要创建许多非叶子节点来构建根节点到该节点间的路径，造成了存储空间的浪费。

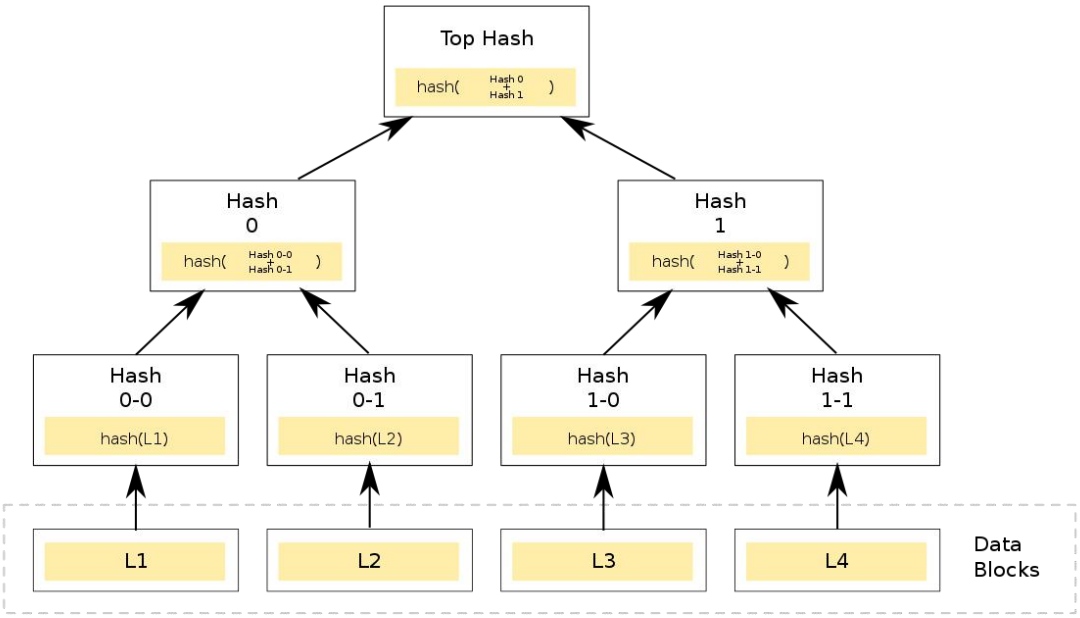
Mercle Tree 的基础概述

Merkle Tree 可以被用于验证任何类型的数据的存储。通常被用作与其他节点的计算机之间进行数据转移的数据完整性以及正确性的校验。

在比特币中。每个区块都有自己的 block header 其中包含了上一个区块的 hash 指针、难度 target 以及 nonce 等信息，最重要的是其中包含的 Merkle root hash。这个信息是当前区块中所有包含的交易信息组成的一颗 Merkle Tree 的根节点的值。有了这个值就可以保证当前区块包含的所有交易信息都不会被改变。

Git 版本控制系统，ZFS 文件系统以及我们自己下载电影常用的点对点网络 BT 下载，都是通过 Merkle Tree 来进行完整性校验的。

Mercle Tree 的构成原理



上图是维基百科的图片，从上图可以看到最底下的一层节点是数据块，对每两个相邻的数据块取 hash 并将它们的值再次进行 hash 得到一个新的节点。再向上将得到的两个相邻的新节点的值做一次 hash 得到一个上层节点。知道最终得到一个根节点。

只需要保存这个根节点的值即可随时验证整个树的所有数据的正确性。因为 hash 的特点，这颗树只要有一个节点的值被更改都会导致最终根节点的值产生

变化。

二叉树的形式，从叶子节点开始向上构建，使用双 SHA256 计算 hash；

如果叶子是单数，则会复制最某位的那个，确保是双数；

因为这些特性，Mercle Tree 的数据块顺序是不可更改的，一旦顺序更改那么所得到的根 hash 值都会不一样。即使两棵树的数据库一样。

Mercle Tree 的特点

Merkle Tree 是一种树，大多数是二叉树，也可以多叉树，无论是几叉树，它都具有树结构的所有特点；

Merkle Tree 的叶子节点的 value 是数据集合的单元数据或者单元数据 HASH。默克尔树的基础数据不是固定的，想存什么数据由你说了算，因为它只要数据经过哈希运算得到的 hash 值。

非叶子节点的 value 是根据它下面所有的叶子节点值，然后按照 Hash 算法计算而得出的。默克尔树是从下往上逐层计算的，就是说每个中间节点是根据相邻的两个叶子节点组合计算得出的，而根节点是根据两个中间节点组合计算得出的，所以叶子节点是基础。

Mercle Tree 的优点缺陷

好处：避免了对所有的交易来进行 hash，可以单独拿分支来对部分数据进行校验，只需要分支就可以计算出 root hash 然后比较，最后还是落到了效率上。

不足：传统的 merkle 树的限制是，它们虽然可以证明包含此交易，但无法证明任何当前的状态，如 bitcoin。因此，以太坊改进使用的是 Merkle Patricia Tree (MPT) ，叫前缀树或者是字典树，具体有待学习。

区块链中的 Merkle tree

Merkle tree 是区块链的基本组成部分。

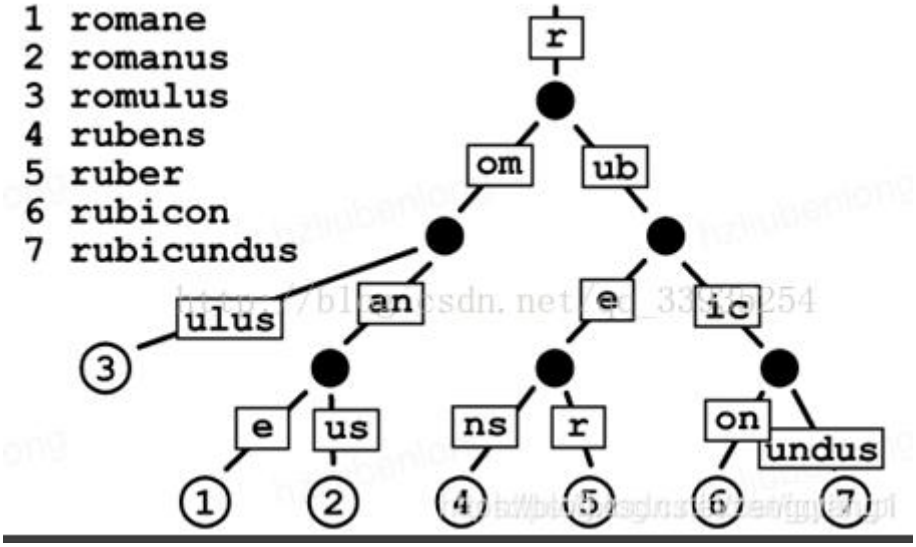
在比特币中，Mercle Tree 主要负责做交易打包的校验，在 block header 中保存了该区块中打包的所有交易组成的一颗 Mercle Tree 的根 hash 值。Mercle Tree 的特性保证了一旦这个区块被链上其他的节点接受，成为最长有效链的一部分之后。这个节点中的交易就不会再被改变，因为一旦改变其中的交易，就会导致整棵树的根 hash 值产生变化，最终当前区块的 hash 值也会改变。这个区块就不会被其他节点接受。

虽说从理论上来讲，没有 Merkle tree 的区块链当然也是可能的，只需创建直接包含每一笔交易的巨大区块头（block header）就可以实现，但这样做无疑会带来可扩展性方面的挑战，从长远发展来看，可能最后将只有那些最强大的计算机，才可以运行这些无需受信的区块链。 正是因为有了 Merkle tree，以太坊节点才可以建立运行在所有的计算机、笔记本、智能手机，甚至是那些由 Slock.it 生产的物联网设备之上。

Merkle Patricia Tree 的构成原理

Merkle Patricia Tree（又称为 Merkle Patricia Trie）是一种经过改良的、融合了默克尔树和前缀树两种树结构优点的数据结构，是以太坊中用来组织管理账户数据、生成交易集合哈希的重要数据结构。它结合了字典树和默克尔树的优点，在压缩字典树中根节点是空的，而 MPT 树可以在根节点保存整棵树的哈希校验

和，而校验和的生成则是采用了和默克尔树生成一致的方式。这是一种更节省空间的 Trie。对于基数树的每个节点，如果该节点是唯一的儿子的话，就和父节点合并。



以太坊采用 MPT 树来保存，交易，交易的收据以及世界状态，为了压缩整体的树高，降低操作的复杂度，以太坊又对 MPT 树进行了一些优化。将树节点分成了四种；

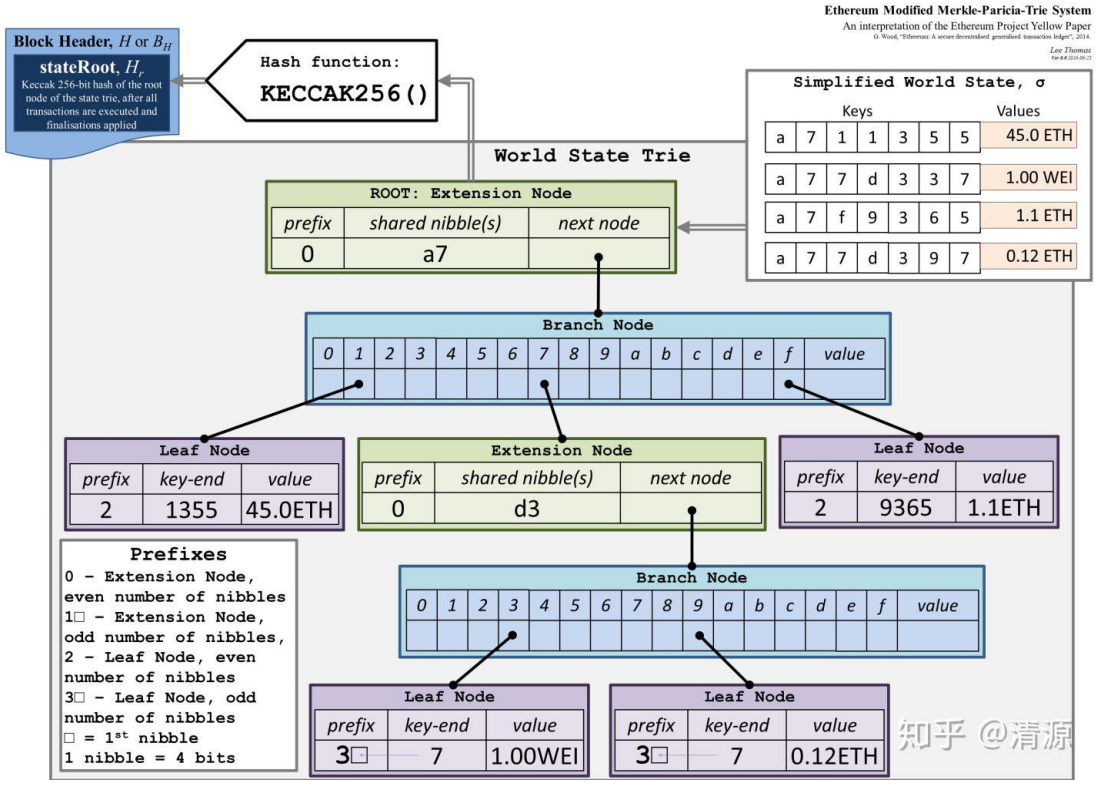
空节点（hashNode）

叶子节点（valueNode）

分支节点（fullNode）

扩展节点（shortNode）

通过以太坊黄皮书中很经典的一张图，来了解不同节点的具体结构和作用



可以看到有四个状态要存储在世界状态的 MPT 树中，需要存入的值是键值对的形式。自顶向下，我们首先看到的 keccak256 生成的根哈希，参考默克尔树的 Top Hash，其次看到的是绿色的扩展节点 Extension Node，其中共同前缀 shared nibble 是 a7，采用了压缩前缀树的方式进行了合并，接着看到蓝色的分支节点 Branch Node，其中有表示十六进制的字符和一个 value，最后的 value 是 fullnode 的数据部分，最后看到紫色的叶子节点 leadfNode 用来存储具体的数据，它也是对路径进行了压缩。

在智能合约执行以后，有一部分数据是需要持久化，参考「EVM 深度分析之数据存储(一)」中的 Storage 中的类型，而一条链上有非常多的合约，每个合约又有很多的数据需要持久化，这个时候就需要用到 MPT 树。

在「EVM 深度分析之数据存储」中可以看到需要持久化的数据都是以键值对的形式存在的，而键是由 keccak256 这个函数计算得到，用十六进制表示后刚好对应 MPT 树中分支节点的 0-f 的 16 个分支。

为了避免不同合约有相同的字段，合约又通过地址来进行管理，具体参考「以太坊账户组织形式」，本质上也是采用 MPT 树管理合约，合约又采用 MPT 树管理自身状态。

最终的数据还是以键值对的形式存储在 LevelDB 中的，MPT 树相当于提供了一个缓存，帮助我们快速找到需要的数据。

Merkle Patricia Tree 的特性应用

1.快速计算所维护数据集哈希标识

这个特点体现在单节点计算的第一步，即在节点哈希计算之前会对该节点的状态进行判断，只有当该节点的内容变脏，才会进行哈希重计算、数据库持久化等操作。如此一来，在某一次事务操作中，对整棵 MPT 树的部分节点的内容产生了修改，那么一次哈希重计算，仅需对这些被修改的节点、以及从这些节点到根节点路径上的节点进行重计算，便能重新获得整棵树的新哈希。

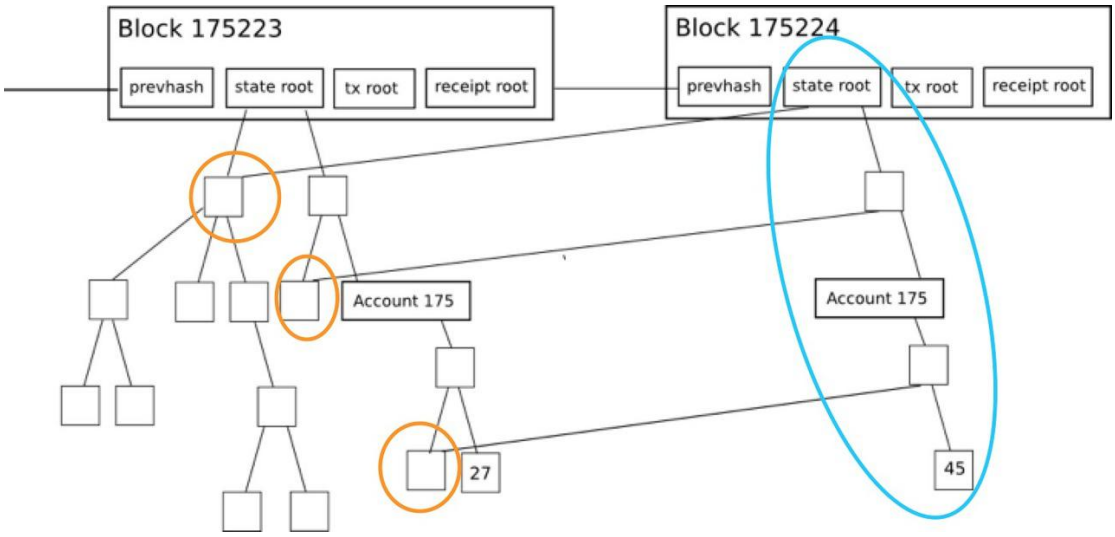
2.快速状态回滚

在公链的环境下，采用 POW 算法是可能会造成分叉而导致区块链状态进行回滚的。在以太坊中，由于出块时间短，这种分叉的几率很大，区块链状态回滚的现象很频繁。

所谓的状态回滚指的是：（1）区块链内容发生了重组织，链头发生切换（2）区块链的世界状态（账户信息）需要进行回滚，即对之前的操作进行撤销。

MPT 树就提供了一种机制，可以当区块碰撞发生了，零延迟地完成世界状态的回滚。这种优势的代价就是需要浪费存储空间去冗余地存储每个节点的历史状态。

每个节点在数据库中的存储都是值驱动的。当一个节点的内容发生了变化，其哈希相应改变，而 MPT 将哈希作为数据库中的索引，也就实现了对于每一个值，在数据库中都有一条确定的记录。而 MPT 是根据节点哈希来关联父子节点的，因此每当一个节点的内容发生变化，最终对于父节点来说，改变的只是一个哈希索引值；父节点的内容也由此改变，产生了一个新的父节点，递归地将这种影响传递到根节点。最终，一次改变对应创建了一条从被改节点到根节点的新路径，而旧节点依然可以根据旧根节点通过旧路径访问得到。



在上图中，一个节点的内容由 27 变为 45，就对应成创建了一条由蓝线圈出的新路径，通过复用绿线圈出的未修改节点信息，构造一棵新树，而旧路径依旧保留。故通过旧 stateRoot，我们依旧能够查询到该节点的值为 27。

所以，在以太坊中，发生分叉而进行世界状态回滚时，仅需要用旧的 MPT 根节点作为入口，即可完成“状态回滚”。

Merkle Patricia Tree 的常用操作

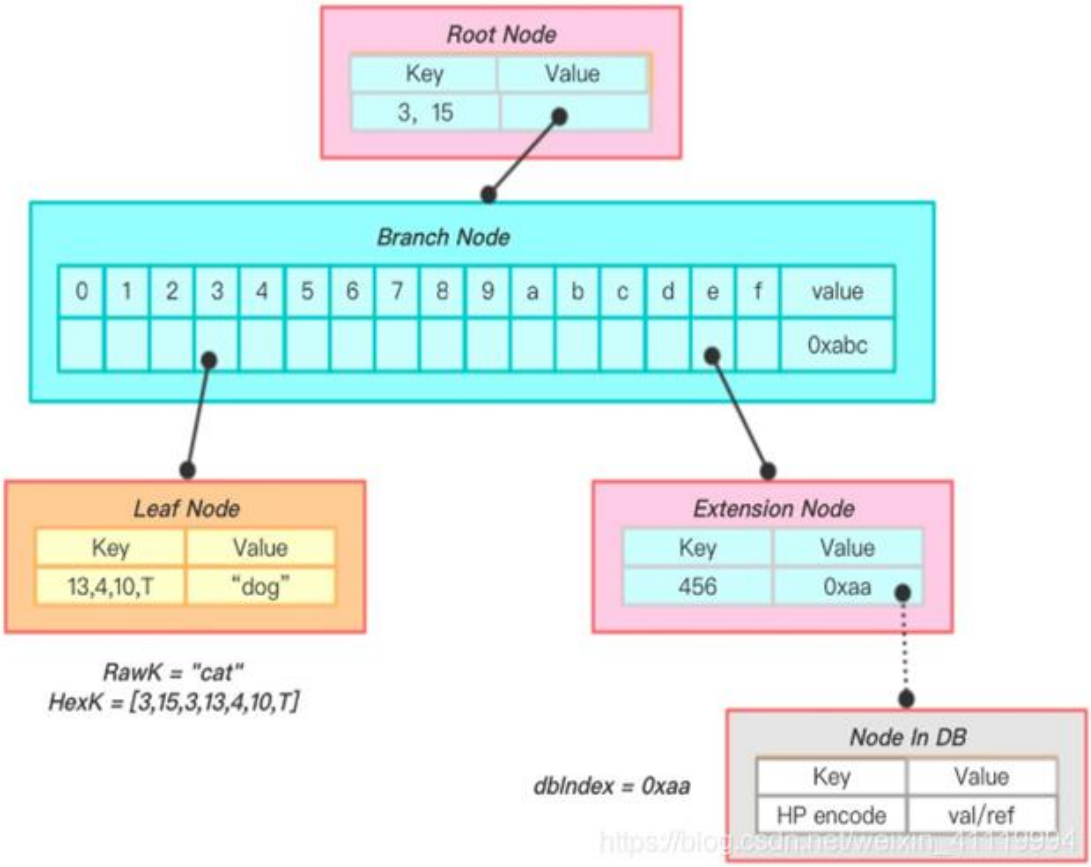
检索

首先将需要查找的 key 转换为十六进制拓展编码，得到的内容就是搜索路径。从根节点开始搜索与搜索路径内容一致的路径：然后分几种情况

1．若当前节点为叶子节点，存储的内容是数据项的内容，且搜索路径的内容与叶子节点的 key 一致，则表示找到该节点；反之则表示该节点在树中不存在。

2．若当前节点为扩展节点，存储的内容是另外一个节点的引用，且当前节点的 key 是搜索路径的前缀，那么就将将搜索路径减去当前节点的 key，将剩余的搜索路径作为参数，对其子节点递归地调用查找函数；若当前节点的 key 不是搜索路径的前缀，表示该节点在树中不存在。

3．若当前节点为分支节点，若搜索路径为空，则返回分支节点的存储内容；反之利用搜索路径的第一个字节选择分支节点的孩子节点，将剩余的搜索路径作为参数递归地调用查找函数



插入

插入操作是基于查找操作完成的，插入过程如图所示：

(1)首先找到与新插入节点拥有最长相同路径前缀的节点

(2)若该节点是分支节点：

1.剩余的搜索路径不为空，那么就将新节点作为一个叶子节点插入到对应的孩子列表中；

2.如果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 Value 中；

(3)若该节点为叶子/扩展节点：

1.剩余的搜索路径与当前节点的 key 一致，则把当前节点的更新即可；

2.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）；