## 3.3 Trie树的插入

下面将会以表格中的数据为基础，详细地展示Trie树插入流程

|  |  |  |
| --- | --- | --- |
| ip地址 | 掩码 | ip信息 |
| 192.0.0.2 | 240.0.0.0 | 1 |
| 96.128.0.2 | 255.192.0.0 | 2 |
| 192.64.0.2 | 255.192.0.0 | 3 |
| 192.128.0.2 | 255.192.0.0 | 4 |
| 96.0.0.2 | 240.0.0.0 | 5 |

1. 首先将十进制ip地址以及掩码转换为二进制形式

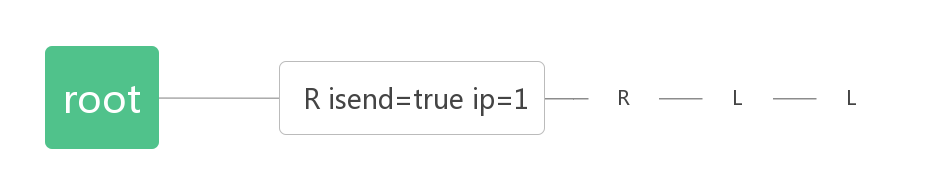
|  |  |
| --- | --- |
| ip地址 | 掩码 |
| 11000000.00000000.00000000.00000010 | 11110000.00000000.00000000.00000000 |
| 01100000.10000000.00000000.00000010 | 11111111.11000000.00000000.00000000 |
| 11000000.01000000.00000000.00000010 | 11111111.11000000.00000000.00000000 |
| 11000000.10000000.00000000.00000010 | 11111111.11000000.00000000.00000000 |
| 01100000.00000000.00000000.00000010 | 11110000.00000000.00000000.00000000 |
| ip信息 | 有效位数 |
| 1 | 4 |
| 2 | 10 |
| 3 | 10 |
| 4 | 10 |
| 5 | 4 |

1. 插入第一个地址：

已知只要插入ip地址的前4位即可完成本轮插入，插入第1位1时发现右节点为空，说明从该节点开始的后续节点都不会与其他ip地址节点有重叠现象，而本节点已是能准确查到该IP的与根节点最近的查询位置，设置其标志位isend为true便于查找时跳出循环，并将ip信息赋给该节点

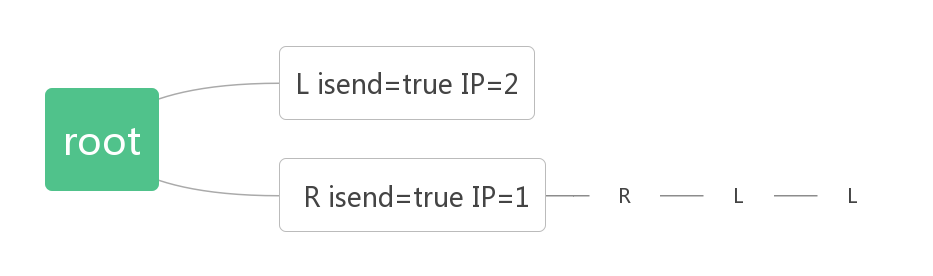


start是已插入位数+1=2，start及之后的ip位（2~4位）直接按顺序插入即可

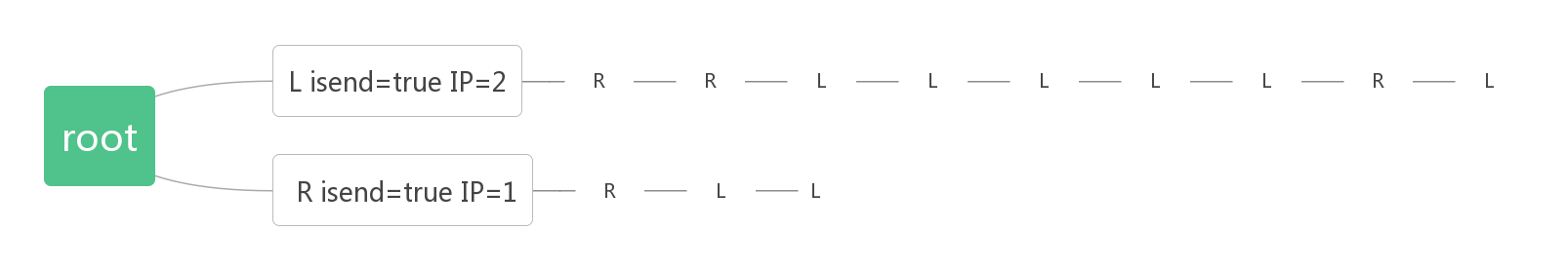


1. 插入第二个地址

已知只要插入ip地址的前10位即可完成本轮插入，插入第1位1时发现左节点为空，说明从该节点开始的后续节点都不会与其他ip地址节点有重叠现象，而本节点已是能准确查到该IP的与根节点最近的查询位置，设置其标志位isend为true便于查找时跳出循环，并将ip信息赋给该节点

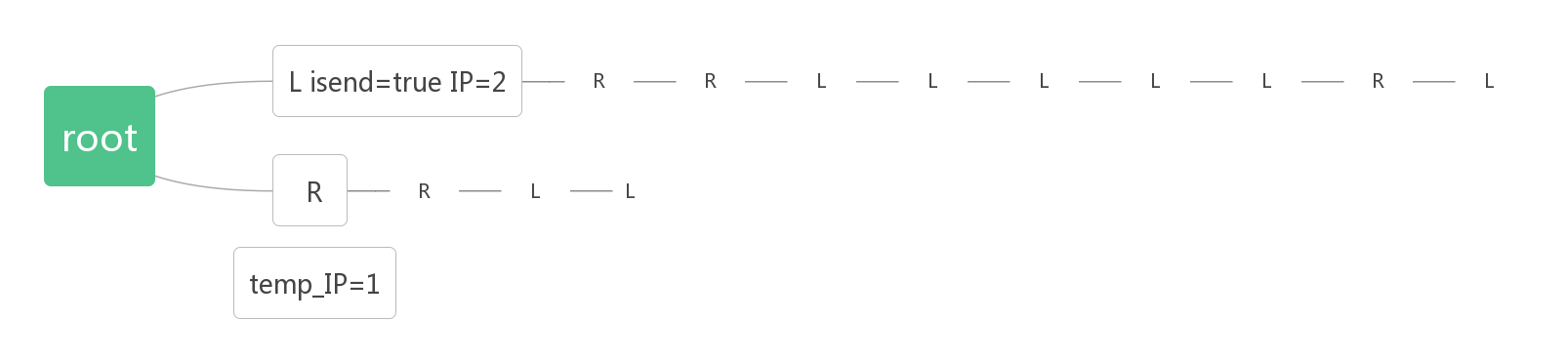


start是已插入位数+1=2，start及之后的ip位（2~10位）直接按顺序插入即可

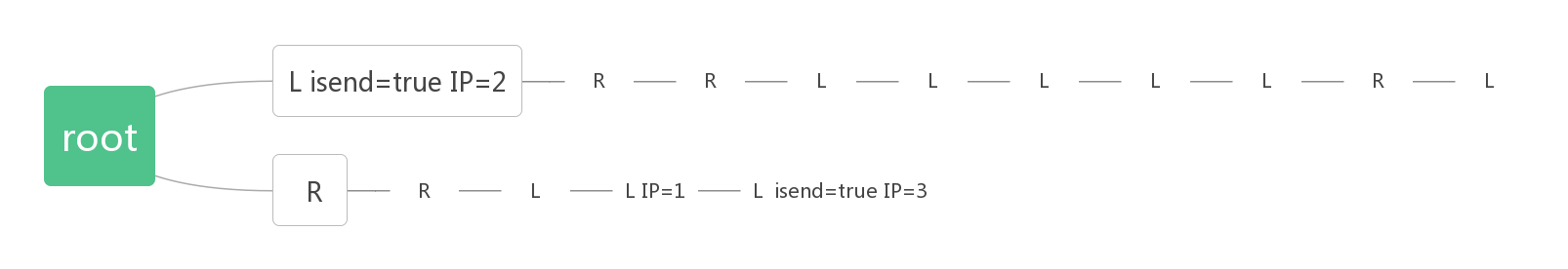


1. 插入第三个地址

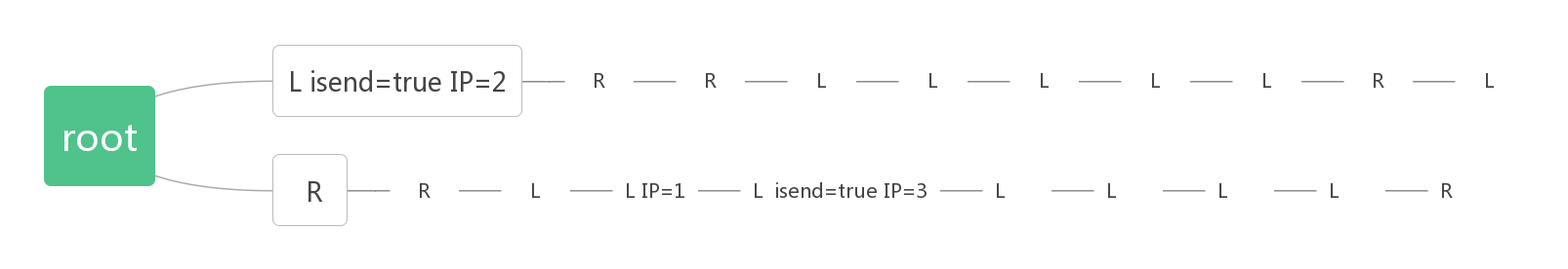
已知只要插入ip地址的前10位即可完成本轮插入，插入第1位1时发现右节点不为空，检查该节点的标志位isend为true，说明其与另一个地址有部分前缀冲突，暂存该节点的ip信息，然后将其删除并将标志位恢复为默认值false，直到在下一分支分离或将前缀全部插入完成的时候重新将暂存的ip信息赋给正确的节点。



现在待插入的第3个二进制ip地址与第1个二进制ip地址前4位完全相同，插入第4位0后发现其左节点为空，说明从左节点开始的后续节点都不会与其他ip地址节点有重叠现象，而左节点已是能准确查到该IP的与根节点最近的查询位置，设置左节点标志位isend为true便于查找时跳出循环，并将ip信息赋给该节点。以此同时其右节点也为空，说明第3个二进制ip地址前缀包含着第1个二进制ip地址前缀，将暂存的ip信息赋给该节点，但并不更改其标志位。

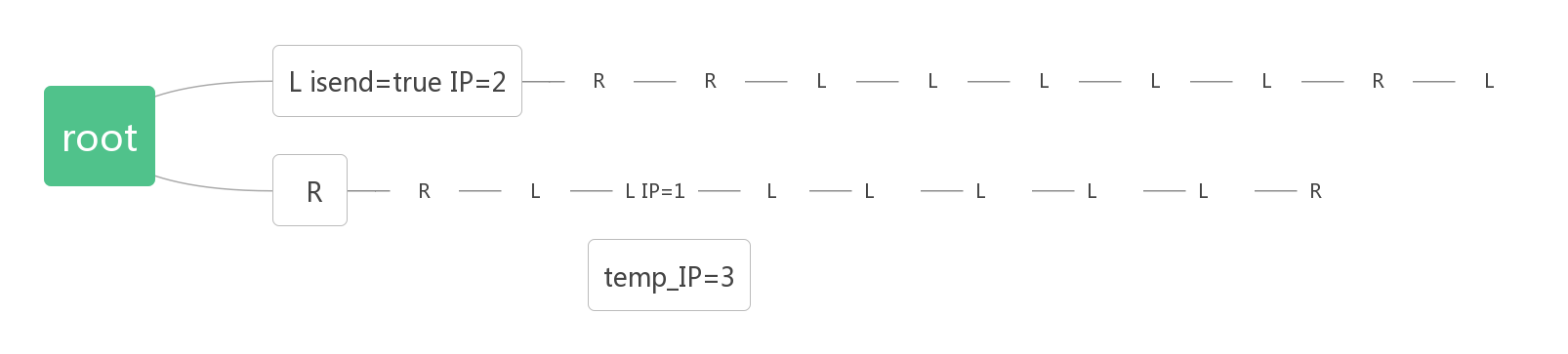


start是已插入位数+1=6，start及之后的ip位（6~10位）直接插入即可

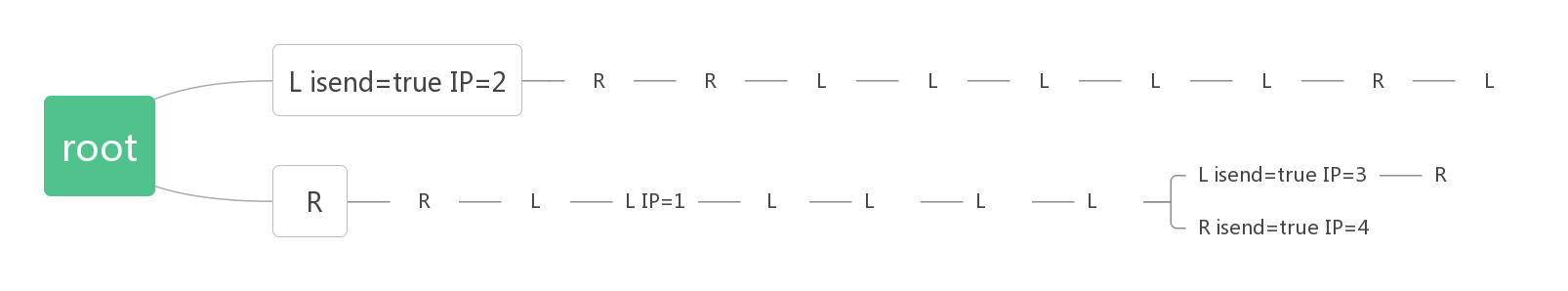


1. 插入第四个地址

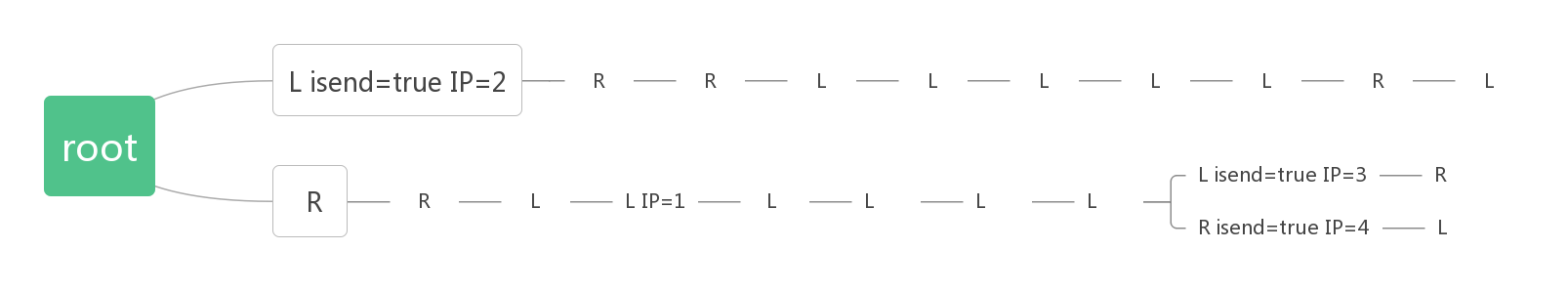
已知只要插入ip地址的前10位即可完成本轮插入，插入第1~4位时发现其比特位对应的左（或右）节点不为空，但是节点标志位为false。插入第5位0时发现左节点不为空，检查该节点的标志位isend为true，说明其与另一个地址有部分前缀冲突，暂存该节点ip信息，然后将其删除并将标志位恢复为默认值false，直到在下一分支分离的时候，并将标志位恢复为默认值false，直到在下一分支分离或将前缀全部插入完成的时候重新将暂存的ip信息赋给正确的节点。



现在待插入的第4个二进制ip地址与第3个二进制ip地址前8位完全相同，插入第8位0后发现其左节点为空，说明从左节点开始的后续节点都不会与其他ip地址节点有重叠现象，而左节点已是能准确查到该IP的与根节点最近的查询位置，设置左节点标志位isend为true便于查找时跳出循环，并将ip信息赋给该节点。与此同时其右节点不为空，说明第4个二进制ip地址前缀与第3个二进制ip地址前缀相交，将暂存的ip信息赋给右节点，设置右节点标志位isend为true。

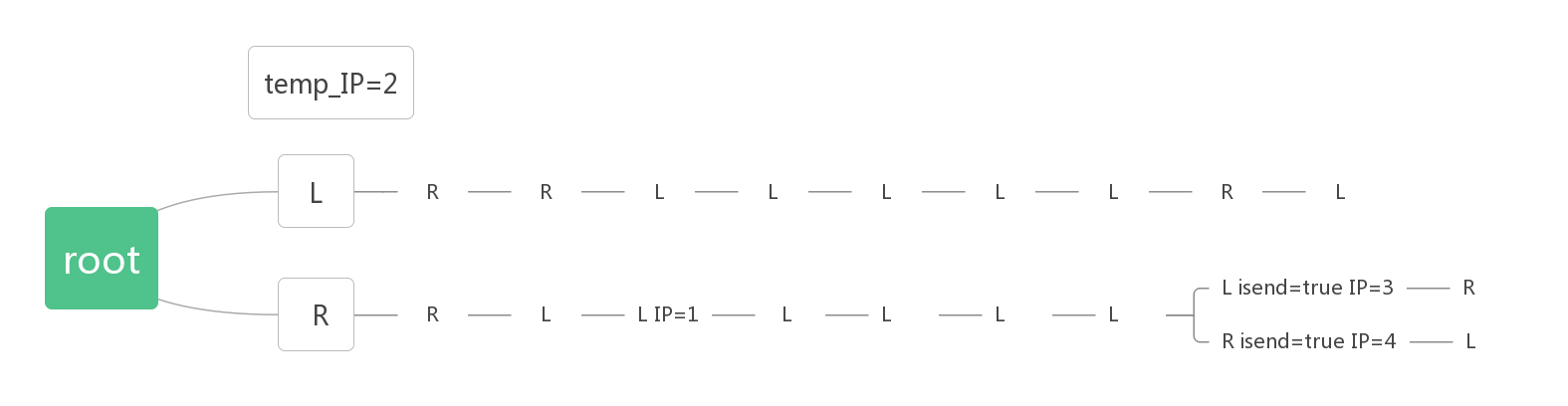


start是已插入位数+1=10，start及之后的ip位（10~10位）直接插入即可

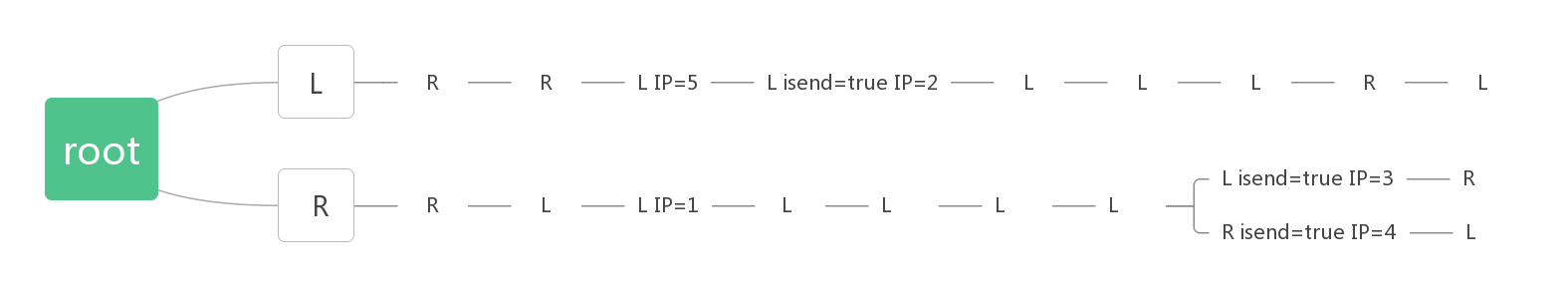


1. 插入第五个地址

已知只要插入ip地址的前4位即可完成本轮插入，插入第1位0时发现左节点不为空，检查该节点的标志位isend为true，说明其与另一个地址有部分前缀冲突，暂存该节点的ip信息，然后将其删除并将标志位恢复为默认值false，直到在下一分支分离或将前缀全部插入完成的时候重新将暂存的ip信息赋给正确的节点。

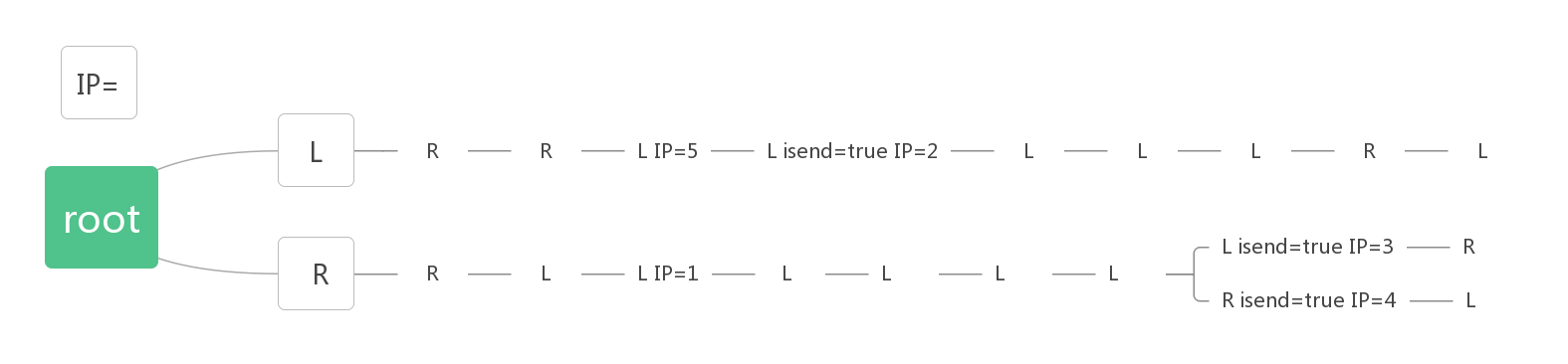


当ip地址的前4位全部遍历完后，并没有发现空节点，则说明第5个二进制ip地址前缀被其他地址前缀所包含，直接将ip信息赋给当前节点也就是第四个节点，但不改变其默认标志位，此时检测到有暂存的IP信息，如果当前节点的左（或右）节点不为空，直接将暂存的IP信息赋给左（或右）节点，并将其标志位isend设为true。

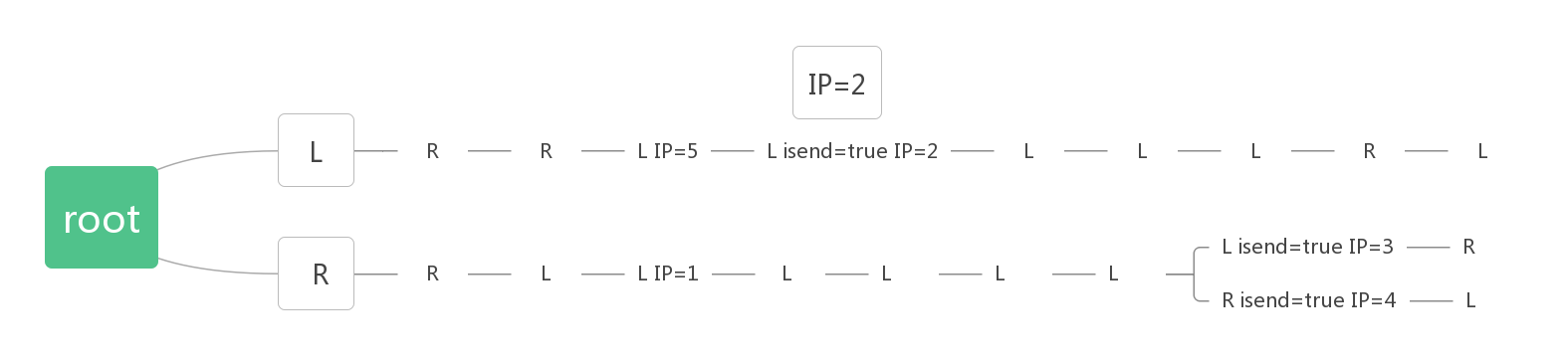


Trie的索引相对来说比较简单，接下来在3.3的工作基础上，以第二个ip地址为例，详细地展示索引流程

1. 将十进制ip地址96.128.0.2转换为二进制形式01100000.00000000.00000000.00000010
2. 正常来说从根节点开始按位查找，当遇到节点标志位为True时该节点的IP信息即为所查的IP地址信息，因为标志节点后面只有空的占位节点



1. 对于现在的情况，当查找到第4位时检测到节点IP=5信息不为空，但是标志位是默认的false，说明前缀之间有包含关系，此时采用最长前缀匹配算法，继续遍历，遇到标志位为true的节点之前，记录最后遇到的IP信息非空的节点所储存的IP信息也就是IP=5，遇到标志位为true的节点之后记录标志位节点信息IP=2，如果能遍历到叶子结点，IP=2即为所查的IP地址信息,否则IP=5为所查的IP地址信息。



由于能遍历到叶子结点，IP=2的前缀为最长匹配前缀