# 专利申请技术交底书

|  |  |
| --- | --- |
| 专利发明人 | 范学鹏 |
| 技术交底书撰写人  (联系方式：电话及邮箱) |  |
| 发明创造类型（发明或实用新型） | 发明及实用新型 |

技术交底书是专利申请的重要依据，请在技术交底书中完整地对技术方案进行说明。请认真填写如下内容：

1. 名称 一种解决流民主问题的方法

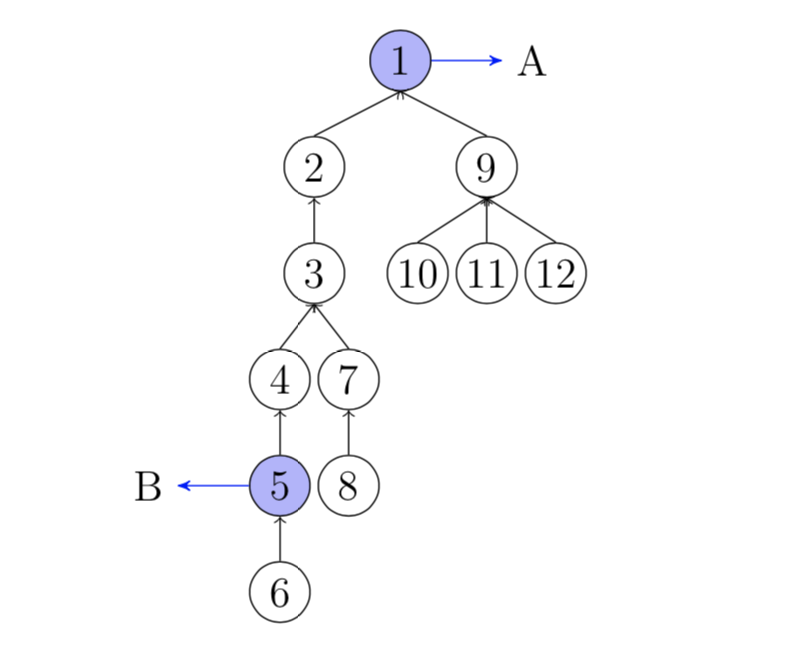
二、所属技术领域 软件工程领域

三、本发明要解决的技术问题是什么？

在现代社会中存在大量涉及人民投票选举/提案的场景。部分科学工作者及爱好者认为，为进一步保证民主性，每个拥有投票权的人都应该对每一次提案行使投票权。然而通常，人们没有那么多的时间参与每一次的提案投票，或者投票者不是提案所涉及领域的专家，无法给出正确判断。这导致大量的投票权实际并没有被行使。

为了解决这个问题，所谓流民主的概念被提出。其核心思想为，每个投票者（称为委托人）可以将自己的票权委托给另一个投票者（称为被委托人），当被委托人投票时，默认将其委托人（包括多级委托人）所有的票权也一起投出。每个投票者的委托人是可以任意指定的，但假设在一次投票提案过程中，每个投票者只能委托一次，且不存在循环委托（即A委托给B，B委托给C，C又委托给A）。

那么，流民主问题中，所有投票者委托关系可以用一个委托图表示，如下图：



图表 1

如图所示，一共有11个投票者，对应图上的11个节点，每个节点所指向的父亲节点为其被委托人。（实际可能存在多个投票者没有委托给任何人（即1号节点），体现为委托图为一个森林，即由多棵不连通子树构成。）

每个投票者拥有一定数量的票权。同时我们假设有若干个候选者，用英文字母A,B,C表示。

为了举例说明，如图1，我们假设每个投票者的票权等于其在图中的编号。当投票者1把票投给候选者A时，根据我们的定义，由于所有的票权最终都汇聚到投票者1，故A的总获票数为1+2+…+12=78。

同时，流民主问题允许任何投票者对自己的被委托人（包括多级委托）投票结果不满意时改变自己的投票，同时被委托人的实际票权相应减少。（以下我们假设每个投票者只能投一次票。对于每个投票者投多次票的情况本发明稍作修改也能解决，但为了方便理解本文不做赘述。）

如图1，当1投完票之后，5再将票投给B，此时B的票数为5+6=11，A的票数相应减少11，变成67。

如果之后投票者3再将票投给候选者C，则C的得票数为3+4+7+8=22，A的票数变为45，B的票数仍为11。

本发明解决的主要技术问题为，基于区块链智能合约实现投票的全过程，包括投票状态的实时显示。即每次输入一个投票者的编号和他投的候选人，我们能够实时输出所有候选者的当前获得的票数。

如上面这个例子：

输入1 A，输出 A 78 B 0 C 0

输入5 B，输出 A 67 B 11 C 0

输入3 C，输出 A 45 B 11 C 22

本发明的一个关键点为，如果投票者的数目为n，我们仅需O(log n)的链上代码执行步数来更新并展示每一次投票,这能够支持以太坊智能合约的gas费支付。同时对于非区块链场景，本发明也能作为一个解决流民主问题的快速算法。

四、背景技术

对流民主问题一直存在广泛的研究讨论[[1]](#footnote-1)。传统的针对链上流民主问题的解决方案都存在各种各样的缺陷。

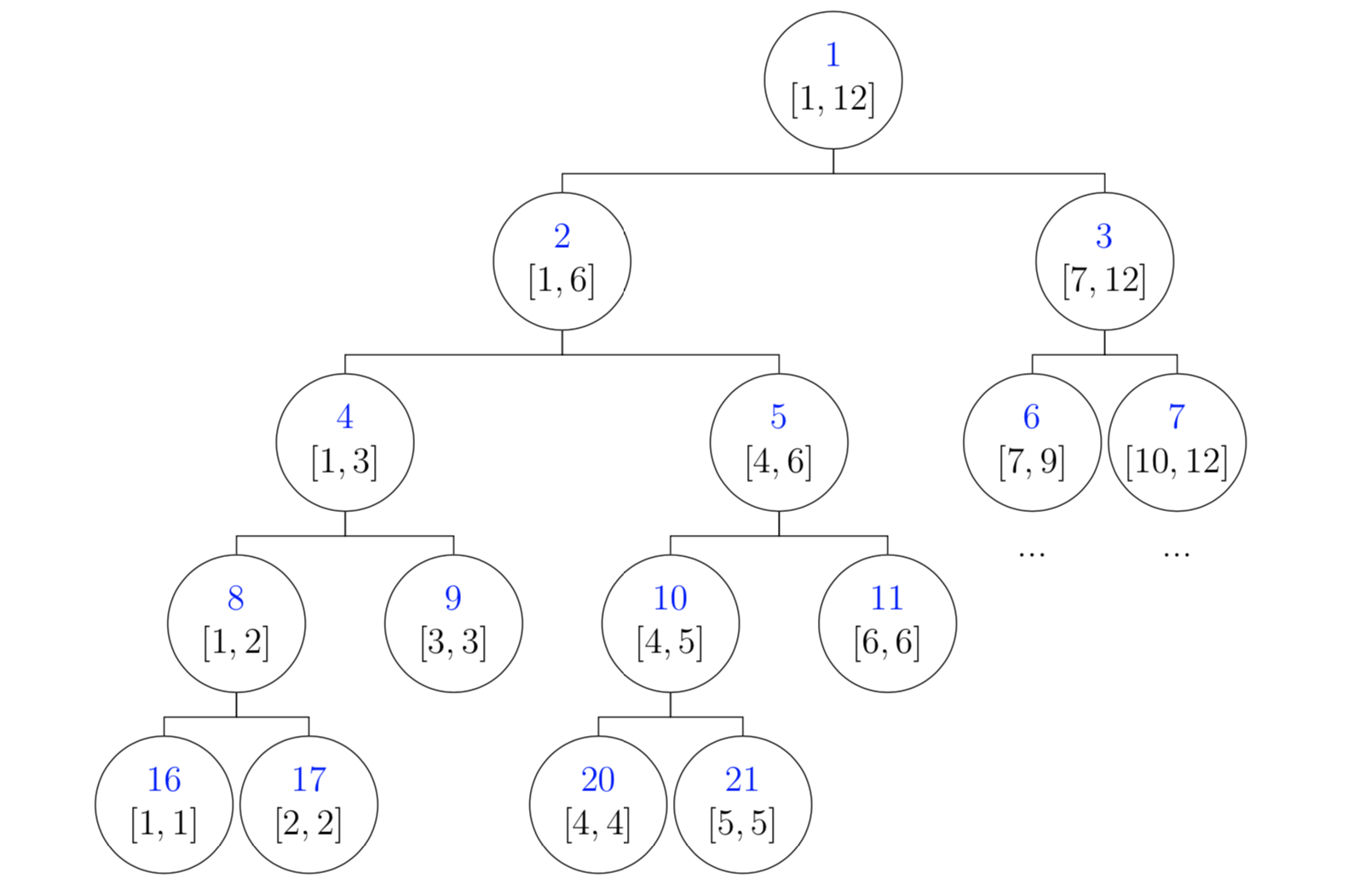
有的论文提出过该问题的链上实现方法[[2]](#footnote-2)，但没有考虑到gas费用的问题：因为投票者的总数目n可以很大(如百万级别)，如果对每一次投票都用最传统的方法，即对图进行一次遍历来计算每个候选者票权的变化，那么链上执行智能合约的时间复杂度将是O(n)级别，而因为链上每执行一步智能合约都要支付gas费，这样gas费的消耗将无法承受。

为了解决智能合约gas费用的问题，有人提出构建委托图时设置最高深度[[3]](#footnote-3)（如100）,即对应委托图中不允许长度超过100的路径出现，这个方案本质上通过限制了用户的行为加了额外约束条件，缺乏对用户的友好性。

还有一些思路是基于构建状态机的方式进行链上运作，即把每次投票当做状态转移的过程。然而这样的状态空间多达n!之多，即使是链下本地操作也无法支持n!时间复杂度的运算。

本发明旨在解决流民主问题链上gas费消耗过多的缺陷，且不增加任何约束条件，用户可以任意进行委托，并且链下的时间复杂度也能承受。

本发明基于的技术主要为两点，一种叫merkel树的链上存储方式和一种叫线段树的数据结构。其中，merkel树为一个平衡二叉树，每个父亲节点的值其子节点值拼接起来做一次（不可逆）哈希函数运算。线段树也是一个平衡二叉树，每个节点对应一个区间，每个父亲节点区间均匀分给两个子节点对应区间，如下图。



图表 2 线段树

其中涉及的技术关键点为

* + - 1. 链上记录merkel树的根节点，每次用户提交merkel树叶子节点数据以及一个merkel 证明(也叫merkel 路径，即对叶子节点数据以及这个路径进行多次哈希运算能够算出根节点的值，长度仅为O(log n))，智能合约能够根据该用户提交的数据以及merkel证明计算出一个值（时间复杂度也为O(log n)），并判断这个值与存在链上的merkel树根节点是否相同，若相同即认为数据合法（数据在merkel树叶子节点里存在）。哈希函数的不可逆性保证用户难以伪造出一个不存在于merkel树叶子节点的数据并通过智能合约检验。
      2. 线段树适用于对一个连续区间进行的调整操作。比如说链上记录了一个长为n的数组s[]，用户每次可能对s[]中的连续几个数进行修改，比如修改s[a-b]。线段树数据结构以及所谓的延迟标记技巧，能保证在O(log n)时间内对s[]进行修改和查找。

五、具体实施方式（应该结合图形详细阐述该技术方案）：

概览：

为了实现投票状态的实时显示，本发明的一个重点为维护并记录所有投票节点的“已损失票权”，初始值为0。那么每次当投票者进行投票时，他的实际投票效用为他的总票权减去他的已损失票权。同时，投票者投完票之后会更新一些其他投票者的已损失票权。只要记录已损失票权的数组能在较短时间内（每次投票O(log n)）成功进行维护，流民主问题就能解决。

继续图表2中的例子：

当投票者1投票给A后，其他所有人的已损失票权不变（仍为0）

当投票者5投票给B后，候选者B实际增加票数为11-0=11（5的总票权减去5的已损失票权）。同时投票者2，3，4的已损失票权均需要加上11，其他未投票的投票者已损失票权不变。

当投票者3投票给C后，候选者C实际增加票数为33-11=22（3的总票权减去3的已损失票权）。同时投票者2的已损失票权需要再加上22，变成33。其他未投票的投票者已损失票权不变。

从图中我们不难看出这么一个结论：每当有投票者投完票之后，需要更新已损失票权的节点为**从投票者到离他最近的投过票的父亲节点的路径上的所有节点**。（注意到在树结构里，每个节点有且仅有一条路径到他某个指定的父亲节点）在这个例子中，B投票时，最近的父亲节点是1，需要更新的路径为(5)->4->3->2->(1)。C投票时，最近的父亲节点还是1，需要更新的路径为(3)->2->(1)。

所以，本发明所采取的算法的关键目标为

1. 找到投票者最近的投过票的父亲节点
2. 更新从投票者到最近投过票父亲节点路径上所有节点的已损失票权。

然而，传统的方法无论实现i)或ii)都至少需要对图的一条路径进行遍历，当图的深度较高时，遍历的时间复杂度仍在O(n)级别，无法满足流民主问题的需求。本发明的解决方案为利用线段树的方式解决，同时用merkel树进行预处理。

接下来算法需要用到两个序列，树的前序序列和括号序列

前序序列，即按照根->叶的顺序遍历树T，图1的前序遍历为1 2 3 4 5 6 7 8 9 10 11 12，即节点的编号和前序序列编号一致

括号序列，仍按照前序遍历，但每次进入和离开节点时均记录一次，图1的括号序列为1 2 3 4 5 6 6 5 4 7 8 8 7 3 2 9 10 10 11 11 12 12 9 1

该方案主要包含三个部分

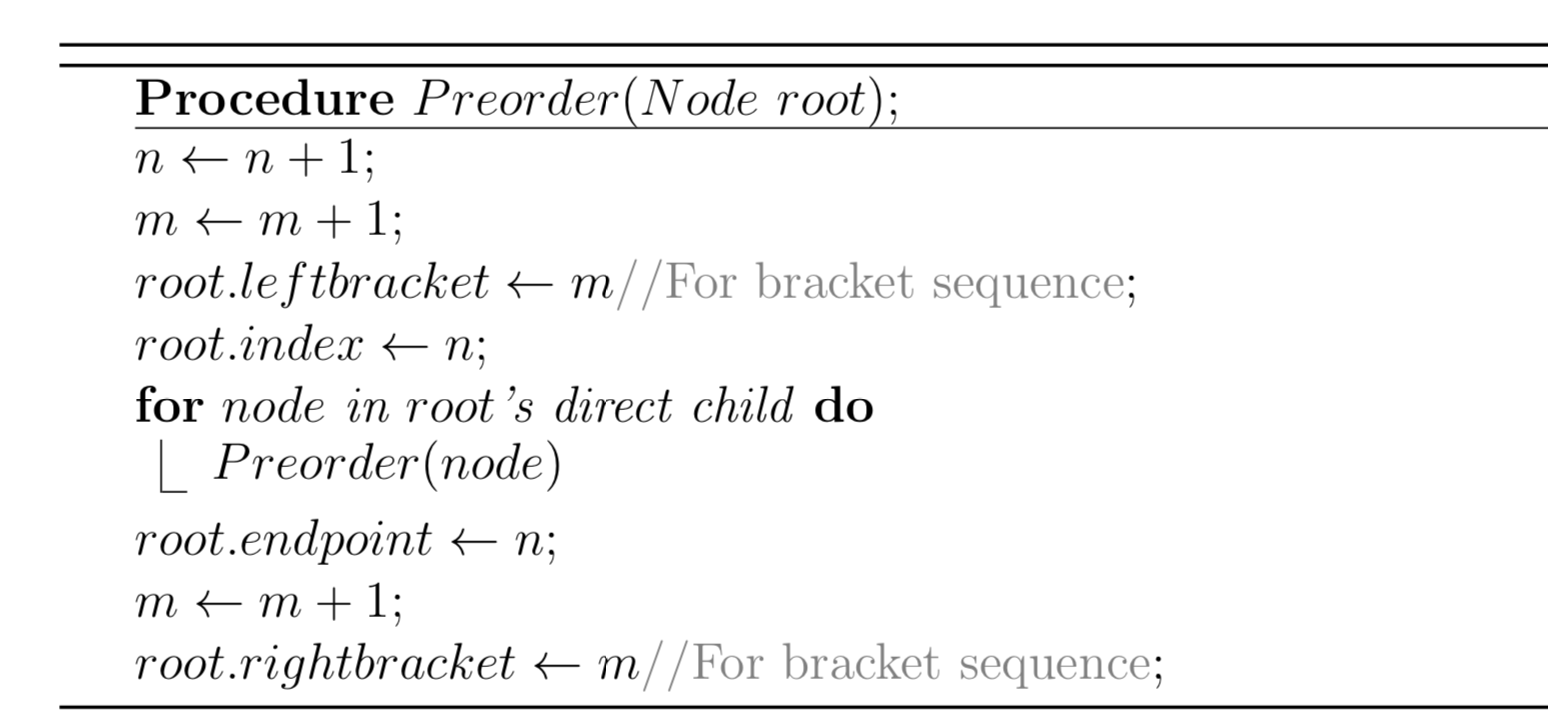
一、投票开始阶段链下对每个参与投票用户的数据预处理，包括他们的实际票权，前序序列编号，括号序列编号（这些后文会用到）等。这些数据均通过merkel树技术在用户发起投票的同时顺便提交到链上。

二、对每次投票行为，查找投票者最近的投过票的父亲节点。该部分通过基于前序序列的线段树来实现。

三、对每次投票行为，更新从投票者到他最近投过票的父亲节点的路径上所有节点已损失票权。该部分通过基于括号序列编号的线段树来实现。

具体步骤如下：

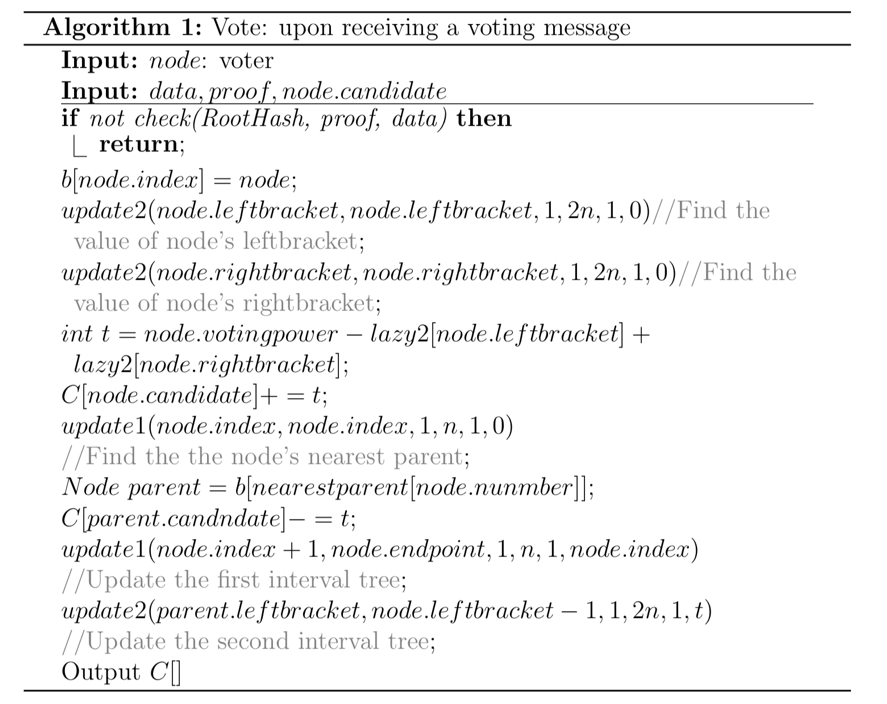
1. 每个投票者提交（或不提交）其被委托人信息，信息收集完毕后，链下建立委托关系图，同时建立一个虚拟节点（编号0），让所有没有出度（被委托人）的节点全部指向这个虚拟节点，这样，整个委托图变成了一棵连通的树T。
2. 给定每个人的票权，投票者本地计算其总票权（node.votingpower，包括其所有委托人的票权），然后对树T进行前序遍历(本地调用Preorder(T.root))，记录所有投票者的前序序列编号（node.index，投票者在前序序列中出现的位置）,子树终端编号(node.endpoint，投票者子树中前序序列编号最大的节点的前序序列编号)，左括号编号（node.leftbracket，投票者第一次出现在括号序列中的位置）,右括号编号node.rightbracket，投票者第二次出现在括号序列中的位置），并记录在data里。



每个投票者根据上述信息构建merkel树，其每个叶子节点对应一个投票者（node），包含的信息为hash(node.adddress, node.votingpower, node.index, node.endpoint, node.leftbracket, node.rightbracket)。所有投票者构建的merkel树应该都是相同的。同时merkel root(根节点)上链存到智能合约中。这个操作可以由此次投票提案的发起者完成。

投票者进行投票操作，发送一个投票信息，包含（data, proof, node.candidate），其中data为该投票者在步骤2本地计算的涉及该投票者的所有数据，proof即为merkel路径，用于判断data是否存在于merkel树上，node.candidate为该投票者投的候选者。

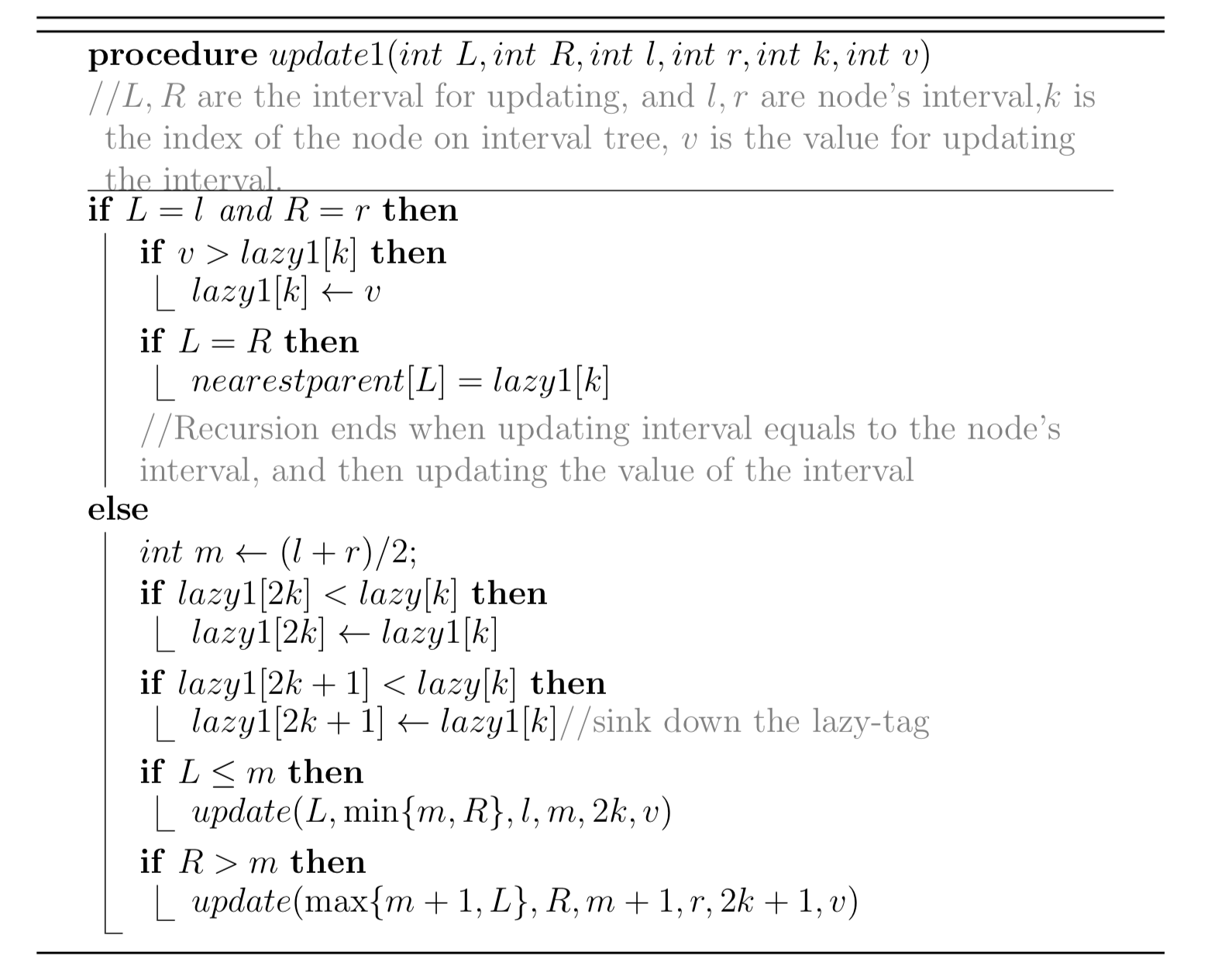
智能合约接收到一个投票信息后，首先获取发送方地址，看是否和data中node.address数据吻合。若吻合，对 data和proof（调用merkel树计算根的流程）计算结果，并判断该结果是否和存在合约里的merkel root吻合，若不吻合返回信息错误。若吻合，智能合约记录一个从用户前序序列编号到节点的映射（即下图的b[node.index]=node）,并开始处理投票信息。



算法 1

以下智能合约交互执行二、三部分。下面分别描述这两个部分。

查找最近的投过票的父亲节点：



上图的算法为针对前序序列构建线段树，并实现更新功能。Lazy1[]数组记录的是一个区间里所有节点（即线段树节点）最近父亲节点编号的最小值。当需要查找一个节点的最近父亲节点时，调用update1(node.index, node.index, 1, n, 1, 0)即可，即对只这个节点的单区间进行更新操作。

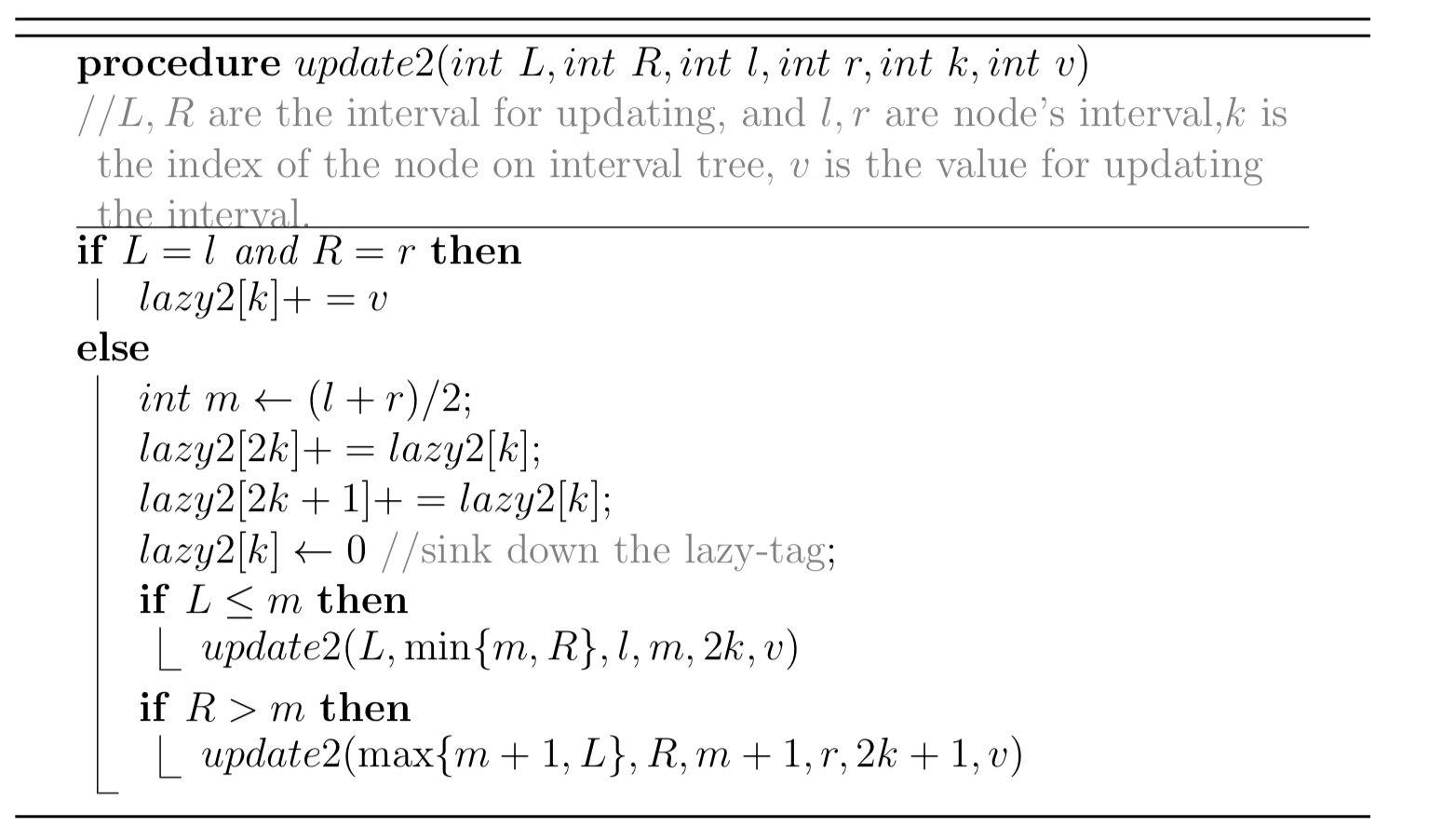
值得一提的是，线段树每次查找操作都要从根节点开始执行一次更新，因为为保证效率每次更新可能只停留到了中间节点，叶子节点还未得到更新，需要在查找时触发lazy标记下沉才能更新到叶子节点。每次查找操作的时间复杂度均为O(log n)。

更新其他节点的最近投过票的父亲节点

当投票者（node）投完票之后，所有需要更新的节点为这个投票者的子节点。而这些需要更新的节点对应的前序遍历编号恰为从node.index+1到node.endpoint,是一个连续的区间，故调用update1(node.index+1, node.endpoint, 1, n, 1, node.index)进行最大值更新即可。（因为在先序遍历中父亲节点的编号总是小于子节点的编号，故对节点进行更新时，判断当前存储在lazy[]中的编号是否小于node.index，若小于则用node.index替换，否则不替换）。

更新路径上（从投票者到离他最近投过票的父亲节点）所有节点的已损失票权：

在图1的例子中，当投票者8投票时，如果8的最近投过票的父亲节点为1，则需要更新的路径为7->3->2->1，在先序遍历序列中不是连续区间，不适用于线段树。



我们的解决方案采用括号序列。上图的算法针对括号序列构建线段树，并实现更新功能。针对括号序列我们定义一个s[]数组，称为s值，初始为0。Lazy2[]数组用于处理线段树区间的s值。

我们定义parent为投票者（node）最近的父亲节点，每次需要更新时，我们让所有从node的左括号编号（node.leftbracket，第一次出现在括号序列的位置）到parent的左括号编号-1(parent.leftbracket-1)这个区间内所有s值加上需要更新的数。这个区间称为路径对应的括号序列区间。在图1的例子中，当8投票时，我们让s[1-10]都加上8（投票者1的左括号编号为1，投票者8的左括号编号为11）。

之后，我们计算节点（node）的已损失票权时，用s[node.leftbracket]-s[node.rightbracket]来计算。

我们不难发现，括号序列满足这么一个性质：假设有一条待更新的路径，比如8->7->3->2->1，对应的括号序列区间为1-10。对于不在路径上的所有点，比如4，5，6，11，12等等，它们区间1-10中或者不出现，或者出现了两次。对于出现在路径上的所有点，如7，3，2，它们恰在对应区间1-10中出现一次。故我们按照上述方式计算已损失票权时，仅在路径上的点增加了已损失票权（其他未在路径上的点的s值或者不变，或者左右括号s值都增加同样的数，互相抵消）。

智能合约处理投票信息总流程：

假设投票信息验证通过，智能合约依次进行如下调用

1. 计算该投票者的已损失票权（分别调用括号序列线段树查找其左右括号的s值，取差值）
2. 将所投候选人票数加上投票者总票权减去已损失票权，用t表示。
3. 查找该投票者的最近父亲节点（对前序序列线段树进行两次调用，一次查找一次更新），用parent表示。
4. parent所投候选者的票数减少t
5. 更新从投票者到parent路径上所有点的已损失票权（调用括号序列线段树，将对应区间所有s值均加上t）

六、有益效果

本方法用O(log n)的链上时间复杂度和O(n)的链下时间复杂度解决了流民主问题。和现有技术相比，

1. 其链上gas费用的消耗为实际可接受水平。
2. 本方法不需要对用户委托行为做任何限制，即用户可以任意指定委托对象。

七、本发明的关键点和欲保护点是什么？

主要欲保护点为针对链上流民主以及链下快速流民主问题的全套解决方案。具体其他欲保护点包括：

1.解决流民主问题的预处理方法，其特征在于，构造merkel树，链上只存储merkel root信息，用户在投票的同时提交链下完成的预处理信息，并能判断正确性，避免了链上gas费消耗

2. 解决流民主问题的线段树方法，其特征在于，将每次查找最近父亲节点转化为连续的子树区间进行最大值更新的问题，并结合线段树数据结构在O(log n)时间复杂度内实现查找和维护。

3. 构造括号序列的方法，其特征在于，将需要更新的路径转化为括号序列的连续区间，使其可以结合线段树数据结构在O(log n)时间复杂度内实现更新。

八、附图

1. <https://techcrunch.com/2018/02/24/liquid-democracy-uses-blockchain/>

   <https://www.redbridge.gov.uk/media/4987/liquid-democracy.pdf>

   <https://liquid.us/> [↑](#footnote-ref-1)
2. <https://skemman.is/bitstream/1946/31161/1/Research-Paper-BBEVS.pdf> [↑](#footnote-ref-2)
3. <https://forum.kleros.io/t/liquid-voting-complexity/71> [↑](#footnote-ref-3)