编程入门作业三

一. 题目简介

分布式数据库往往由多个节点（Node）组成，每个节点会存储不同的数据，我们会根据每个节点的硬件条件（如 cpu 性能，内存大小）来分配不同的功能，进而来使得整个系统效率最高。每个节点，我们会结合其功能和数据分布，选择其对应的数据结构。本题中我们将考虑一个简化的情况。

假设分布式数据库系统中有 m 个计算节点 (Node_id：[0,m-1])，n 个存储节点 (Node_id：[0,n-1])。计算节点的作用是与用户交互，响应用户的；存储节点的作用是存储数据，每个节点会被唯一的 Node_id 标识。**整个系统对用户提供四个接口：Insert，Query，Update 以及 GetNode。**所有数据都由一个可以唯一表示该数据的 key 和一个 value 组成，key 和 value 都是整数。

全部指令都会首先经由计算节点，由用户指定执行操作的节点 id，由对应的节点执行指令，如果用户未指定，系统会选择目前已经执行操作数量最少的且闲置的计算节点来响应用户请求（初识时所有节点执行操作数都是 0）。每个计算节点会缓存其最近访问过的 k 个数据（无论任何操作访问过都算）。(即如果 insert 但是 exist，则 insert 的 key 算是最新访问过，如果 query 是 not found，cache 里也要存一下这个 key 并记录它 not found，如果 cache 中的 key 被更新缓存或者被查询，也是最新访问过)

- 对于 **Insert** 操作，我们输入一个 key (整数，可唯一标识一个数据) 和一个 value (整数)，由目前执行操作数最少的计算节点（如果操作数相同，则选择 id 小的 node）来执行，如果指定了处理指令的计算节点 id，则由相应的节点执行。其会计算 id=key%n 通过来得到该数据应该被插入的存储节点 id，交由相应的存储节点进行处理。如果该 key 已经存在，则输出“EXIST”。如果先有的 value 与插入的 value 不一样，也不修改现有 value，但是要在缓存中标记为最新访问过。
- 对于 **Query**，我们输入一个 key，由已经执行操作数最少的计算节点进行处理，同样计算 id=key%n 来到对应的存储节点查询数据，随后我们输出 value。如果该 key 并不存在，则输出“NOT FOUND”。**如果该 key 在当前计算节点有缓存，则不需要搜索存储节点。**
- 对于 **Update**，我们输入 key，value，同样采用上述办法计算出对应的存储节点 id，修改对应的数据，分别输出原来的 value 和新的 value，需要注意的是如果这个 key 被其他计算节点缓存，则需要同步修改其他计算节点的缓存（按照从 Node_id 从小到大的顺序）。如果该数据本不存在，则插入该 KV pair。

对于存储节点，其会根据数据规模不断调整存储数据所用的数据结构，为简化模型，我们假设有数组（Array），二叉树（Binary Search Tree），和二叉 Trie 树（Trie）三种存储结构。

- 当该节点的数据个数小于 64 的时候，我们采用数组来存储数据，每次将数据添加在数组末尾即可。
- 如果某一次插入操作后，该节点数据个数大于等于 64。我们将该有序数组转换成二叉搜索树来提高性能。二叉搜索树是指对树中的任意节点，其左子树上的节点里的 key 都比该节点中的 key 小，其右子树上的节点里的 key 都比该节点中的 key 大的二叉树。在转换的过程中，我们将数组从头到位插入到空树里面。

- 如果某一次插入操作结束后，发现该节点的二叉搜索树存在一条长度（从根节点到叶子节点的节点数）大于等于六分之一该存储节点数据总量的路径，则将该存储节点的存储结构变为二叉 Trie，关于 Trie 树的简介见 https://zhuanlan.zhihu.com/p/28891541。

二. 输入输出

首先输入一行，分别是 m，n，k，N，中间以空格隔开，其中 N 为后面的操作数。M，n，k 都小于 8，N 小于 1e6

随后输入若干行，包含总共 N 条指令，其中时间戳不算指令，输入格式如下

- TimeStamp //一个时间戳，表明下面的指令发生在这个时间，直到出现下一个时间戳。输入的时间戳是递增顺序。
- Insert key value (ComputingNodeID)
- Query key (ComputingNodeID)
- Update key value (ComputingNodeID) //我们认为如果一个节点没有缓存这个 key，在 update 的时候就不需要被访问，尽管代码逻辑上可能需要搜索。
- GetNode Node_id (ComputingNodeID) //Node_id 保证是存储节点的 id

我们认为同一个时间戳下输入的指令按照输入先后执行，每个计算节点在一个时间戳下只能被访问一次，如果该节点在当前时间戳下已经被分配了指令，则该节点无法继续执行其他指令。如果有用户指定由一个当前正在执行其他指令的计算节点营业的话，则输出“REJECT”。一个特殊的情况是修改缓存，如果该节点正在修改缓存，则当前时间戳下无法继续营业，但是如果该节点在执行其他指令，此时需要修改缓存，则可以正常营业（background 策略）。注意修改缓存不增加操作数，只有响应用户请求的时候（insert，query，update，getnode）需要增加操作数。

每次无论何种原因访问一个节点，都要首先输出“Visiting Computing/data node $node_id at time $Timestamp”并换行。每种操作对应的输出如下，每次输出都要换行。

- Insert：插入成功则输出“Successfully Inserted”如果已经存在则输出“EXIST”
- Query：查询成功则输出 value。如果不存在，则输出“NOT FOUND”
- Update：key 存在则输出 old value 和 new value，中间用空格隔开，否则此指令等同于 Insert。由于更新的过程中存在缓存问题，我们在更新了存储节点之后才会去更新计算节点当中的缓存。
- GetNode：模拟真实场景中的数据传输过程，在本题中我们简化问题，每次 GetNode 会先访问一次计算节点，再访问 Node_id 对应的存储节点，我们将输出对应存储节点当中的数据结构。
  - 如果该存储节点当中使用的是数组，我们输出一行“Array”并换行，随后输出一行为该数组的 key（不输出 value），每个 key 以空格分开。
  - 如果使用的是树结构，为了避免树结构中过多的指针被传输，我们输出该树的 succinct 表示，来减少数据传输。第一行为该树的类型（Binary Search Tree/Trie）。第二行为该树的层次遍历序列，以空格分隔。第三行为层次遍历序列下树的每个节点是否是某个节点的第一个孩子（是的话输出 1，不是的话输出 0），第四行为该节点是否是叶节点（是的话输出 1，不是的话输出 0）