Search By Name ID Algorithm for the Skip Graph

Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü and Öznur Özkasap
Department of Computer Engineering, Koç University, İstanbul, Turkey
{yhassanzadeh13, akupcu, oozkasap}@ku.edu.tr

I. Algorithm Overview

The *search by name id* algorithm receives a target name id as a binary string, searches for it through the Skip Graph and returns the address of the node holding that name id. If the node who holds the target name id does not exist in the Skip Graph, the search algorithm returns the address of one of the nodes holding the most similar name id to the search target. The name ids similarity is defined by the *common prefix length* of them. The longer common prefix two nodes have in their name ids, their name ids are more similar to each other.

The *search by name id* is a distributed recursive algorithm where each recursion continues on a separate node. The basic idea of the algorithm is to find the node with the name id that has the longer common prefix with the search target than the current level number of the search. The algorithm then jumps to the corresponding level and continues the search in that level in the same manner recursively. The search is started by the node who initiates the search from a certain level. That certain level number corresponds to the common prefix length in the name ids of the search initiator and the search target. The search is terminated when either the search target has been found or when in a certain level no node is found with common prefix with the target name id greater than that level number.

As an example, in Figure 1, there is no node with the name id of 010. The most similar name id to 010 in the Skip Graph is 011. This is the name id of the node holding the numerical id 55. The 011 and 010 name ids have two bits prefix in common. Both of their name ids start with 01 prefix. If the node with the numerical id 55 and name id 011 does not exist in the Skip Graph, the most similar name ids to the target name id 010 are 000 and 001. These name ids both have only one bit prefix in common with the target name id.

Similar to the search by numerical id, the *search by name id* can be initiated and performed by any node of the Skip Graph. Furthermore, an external node that does not belong to the Skip Graph can initiate this search via an internal node of the Skip Graph.

II. Algorithm Description

a) Inputs and Output: Algorithm I.1 shows the *search by name id* procedure that receives two pointers $Left$ and $Right$, the search target name id as a binary string ($searchTarget$) and the current level number ($Level$). It returns the address of the node who holds the target name id as the $Result$ pointer.

Algorithm I.1: Search By Name ID

Input: pointer Left, pointer Right, String searchTarget, int Level
Output: pointer Result

```
1  pointer Buffer = Null;
2  while commonBits(searchTarget, Right) <= Level AND
   commonBits(searchTarget, Left) <= Level do
3      if Left.nameID == searchTarget then
4          return Left;
5      if Right.nameID == searchTarget then
6          return Right;
7      if Left ≠ NULL then
8          Buffer = Left;
9          Left = Left.lookup[Level][L];
10     if Right ≠ NULL then
11         Buffer = Right;
12         Right = Right.lookup[Level][R];
13     if commonBits(searchTarget, Right) > Level then
14         Level = commonBits(Right, searchTarget);
15         Left = Right.lookup[Level][L];
16         Right = Right.lookup[Level][R];
17         SearchByNameID(Left, Right, searchTarget,
           Level);
18     else if commonBits(searchTarget, Left) > Level then
19         Level = commonBits(Left, searchTarget);
20         Left = Left.lookup[Level][L];
21         Right = Left.lookup[Level][R];
22         SearchByNameID(Left, Right, searchTarget,
           Level);
23     if Left == NULL AND Right == NULL then
24         return Buffer;
```

We assume that each Skip Graph node has a lookup table in the form of a two dimensional array defined as $lookup[levels][2]$, where $levels$ is the number of Skip Graph's levels that is equal to $\lceil \log N \rceil$ in a Skip Graph with N nodes. For a certain node, $lookup[i][R]$ and $lookup[i][L]$ return the right and left neighbors of the node in the i^{th} level of the Skip Graph, respectively.

Assume that the node α wants to perform a search for the $target$ name id. Then, it initiates the search by calling the $SearchByNameID(\alpha.lookup[cpl][L], \alpha.lookup[cpl][R], target, levels)$, where *cpl* is equal to the common prefix length in the name ids of the search initiator and the search

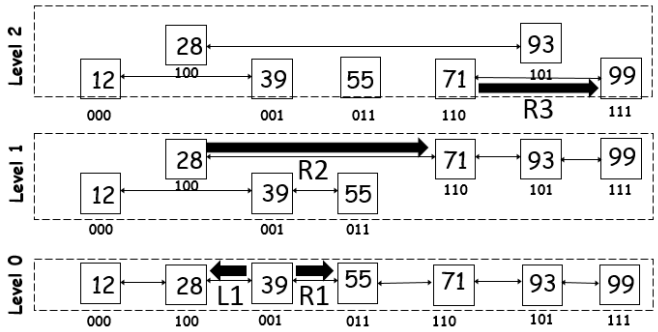

Fig. 1: An example of the *search by name id* algorithm. The search initiator is the node with name id 001 and search target is the node with name id 111

target.

b) Searching in a list (Lines 2-12): In the Algorithm I.1, in a certain level, while neither the search target has been found nor a node who holds common prefix length with the search target greater than the number of that level, the search will be continued by the $Left$ and $Right$ pointers in the left and right directions in that level concurrently. When each of the $Left$ or $Right$ pointers reach the left or right end of the list in a certain level, respectively, their values become $NULL$ and they can not go any further.

c) Jumping to the upper level (Lines 13-22): While the $Right$ and $Left$ pointers traverse a list in a certain level, if they find a node that has greater common prefix in its name id with the target name id than the current level number, the search in that level is terminated. The algorithm then jumps to the level that corresponds to the length of the common prefix in the name ids of that node and the search target, sets the pointers to their new values and continues the search recursively.

d) Returning one of the most similar name ids (Lines 23-24): There may be a case when both $Right$ and $Left$ pointers reach to the right and left end of a list in a certain level. In such case, there is not a more similar name id to the search target in the Skip Graph. After a jump to the upper level, all the nodes in the new level have the most similar name ids to the search target up to that point of the algorithm execution. Reaching both ends of the list in a certain level means that there is no node with a better similarity to the search target. In this situation, the most similar existing name ids to the search target are the ones in the current level. All of the nodes in the current level have the common prefix in their name ids with the search target equal to the number of the current level. To return the most similar name id as the result of the search, it is enough to select one of the nodes in the current level. The algorithm performs this task by keeping the latest value of the $Right$ and $Left$ pointers in the $Buffer$ pointer at the time their value is going to be changed (Algorithm I.1, Lines 8 and 11). When both $Left$ and $Right$ pointers reach the end of a list, the algorithm returns the value of the $Buffer$ pointer as one of the most similar name ids to the search target.

III. Example

Figure 1 shows an example of the *search by name id* algorithm, where the node with name id 001 and numerical id 39 initiates a search for the target name id 111. The Rx and Lx notation corresponds to the values of the $Right$ and $Left$ pointers during the execution of the algorithm. Since the common prefix length of 001 and 111 is $zero$, the search is started at the level $zero$. The initiator sets the $Left$ and $Right$ pointers to its left and right neighbors in the level $zero$, respectively. The name id of the node who L1 holds its address has one bit common prefix with the search target which is greater than the current level number ($zero$). Therefore, the initiator passes the search to the node with numerical id 28 and name id 100, and the algorithm jumps to the level 1. In the level 1, the common prefix length between the name id of the node that R2 holds its address and the target name id is 2 bits which is more than the current level number. Therefore, the search is passed to the node with numerical id 71 and name id 110, and the algorithm jumps to the level 2. In the level 2, R3 holds the address of the node that has the search target name id (111). The search is therefore finished and the value of R3 is returned.

IV. Time Complexity

As was described earlier, the main operations of *search by name id* algorithm are jumping to the upper level and scanning there recursively.

Theorem 1: In the *search by name id* algorithm, with high probability, number of the scan operations in a certain level is $O(1)$.

Proof: In the k^{th} level, the search continues in the left and right directions until a node is found such that its common prefix length with the search target is longer than k bits. Based on the logic of the search algorithm, all of the nodes placed in same list in the k^{th} level have at least k bits common prefix with the search target. Without loss of generality, assume that the $k+1^{th}$ bit of the search target name id is $zero$. Therefore, the search in the k^{th} level continues until a node is found such that its $k+1^{th}$ bit of name id is equal to $zero$. Starting from a certain node in the k^{th} level, the probability that after scanning a single node in both directions, no node is found having the $k+1^{th}$ bit equal to $zero$ is $\frac{1}{4}$ (The probability is equal to $\frac{1}{2}$ for each direction). The probability that such a node *is not found* after a *constant* number of scan operations in both directions is equal to the probability that such a node *is found* after a *variable* number of scans, say m scan operations. The probability that such a node is found after m scans in both directions is equal to $(\frac{1}{4})^m$. Based on the above discussion, the probability of having the constant number of scan operations in a certain level is equal to the complementary probability of terminating the scan operations after scanning m nodes in both directions which is equal to the $1-(\frac{1}{4})^m$.

Theorem 2: In *search by name id*, having N nodes in the Skip Graph, the number of jumps to the upper level is $O(\log N)$.

Proof: Number of the jumps is bounded by number of the levels in Skip Graph which is $O(\log N)$.

Theorem 3: In a Skip Graph with maximum N number of nodes, the *search by name id* traverses $O(\log N)$ nodes on average to perform a search for a name id.

Proof: Based on the Theorems 1 and 2, in the worst case, the *search by name id* algorithm traverses all the levels one by one. In each level the number of the scan operations is $O(1)$. Therefore, the asymptotic running time of *search by name id* is bounded by the number of Skip Graph's level which is $O(\log N)$.