Improving Shin's Massive Cross-Referencing with Shin Sort

Printed on Nov. 6, 2007

Dr. Dong-Keun Shin

DK Shin Laboratory
Hwa Shin Building, Suite 701
705-22 Yuksam-dong, Kangnam-gu
Seoul, Republic of Korea

Abstract

The author of this paper, Dong-Keun Shin combines his method for massive cross-referencing or equi-join database operation with his sorting method which is called, "Shin sort." Shin sort helps the author's massive cross-referencing method by removing the time in scanning through source and target hash tables. The author's massive cross-referencing method can be improved by replacing hash tables with source and target Shin trees that are created in the process of Shin sort. However, the stack oriented filtering technique(SOFT) that the author uses in his original join method stays the same. Five functionally different hash coders and a stack will still be used in dividing source list and target list and in keeping track of their respective sublist. By the Shin Sort the SOFT will be enhanced in terms of speed, and then it will be used in S^3-DBMS.

Introduction

The author's doctoral dissertation in 1991[1] and his subsequent papers[2,3] explain enough about his join database algorithm. After the author discovered a new sorting and searching method in 1998, he realized that his new sorting method can also improve his join method. This paper will show how his sorting method improve his method for the massive cross-reference or the join.

To understand the Shin sort, one needs to solve several problems based on the author's explanation on his algorithm. DK Shin has given sample problems in his papers[4,5,6] in 1998. He also used the examples in writing his program for Shin sort. The program has successfully worked

Copyright© 2007 by Dong-Keun Shin. All rights reserved.

because he has exercised enough with the sample problems for implementing his sorting method. Here one of the examples will be used. The input list that is written in the author's July 4, 1998 paper[4] is used for the source list that contains nine input strings. Target list contains eight strings for last name. Let's assume that for each string the first hash coder provides a hash address that is greater than or equal to zero and smaller than or equal to 255. We put a string with its generated hash address like KIM(87). This means that the first hash coder reads input string "KIM" and it generates 87 for its hash address. The number 87 is an acceptable hash address because $0 \leq 87 \leq 255$. The source and target list and their corresponding Shin tree are shown in figure 1.

Source List: KIM(87), KING(123), LION(204), KIND(31), JADE(64), KIN(208), KENT(17), KILE(196), QUEEN(123)

Target List: SHIN(251), SMITH(157), MOHAMAD(172), KIM(87), WANG(196), BROWN(22), LEE(138), KING(123)

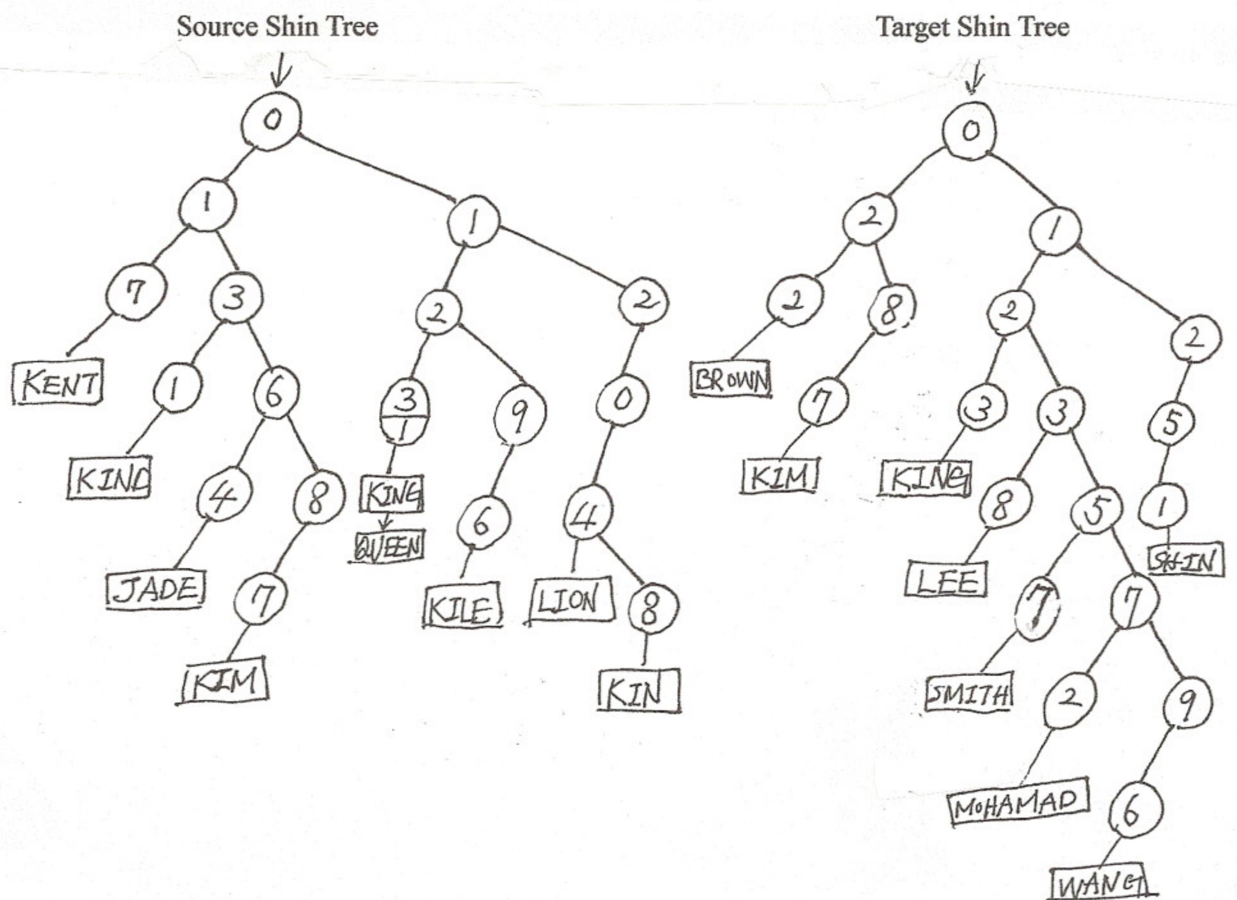

Figure 1: An Example of Source and Target List and Their Corresponding Shin Tree

The example shows how the source and target strings are inserted into their corresponding Shin tree after the keys are hashed by the first hash coder. As the both trees show, the terminal nodes of the trees should point to the item that is the key and its associated information. The above example has only one field, which is the key itself, in each item. Two or more than two keys may have same hash address such as "KING" and "QUEEN" in the source list. Their hash address is 123. As shown in the source Shin tree, both are linked together from the terminal node. The linked list is sent to another hash coder for further division in most cases.

Combining Shin's Massive Cross-Referencing with Shin Sort

The figure that appears in my papers and helps readers to understand the stack oriented filter technique needs to be modified as shown in figure 2 in this paper. The Shin sort is adapted for the massive cross-referencing to suit increasing speed in the process of the SOFT. The hash tables in old paper [1] such as S, T, Si, Ti, Sij, Tij, Sijk, Tijk, Sijkl, and Tijkl are no longer needed. However, the stack is still desired to keep track of return bucket address in both source and target Shin tree. Instead of source hash table and its corresponding target hash table, the source Shin tree and the target Shin tree are recommended to be used to speed up the filtering process. Replacing the source and target hash tables with the source and target Shin trees is the only change and all other ideas stay the same in the algorithm.

While the source and target lists are being repeatedly divided by a maximum of five functionally different hash coders, detected unnecessary items are eliminated and a group of source and target items are merged if they have an identical key. The SOFT examines produced hash addresses in both source and target Shin tree, traversing the trees in preorder. Since each node contains a digit for hash address number, the traversed nodes' digits will be pushed into a stack one by one. Being pushed into and popped from the stack, the digit number of the traversed nodes will provide a hash address. Then the SOFT compares the hash address provided from the source Shin tree with its closest hash address in the target Shin tree. If both hash addresses are identical, the items in the linked list of the source Shin tree and the items in the corresponding target Shin tree for the hash address have potential to be included in the resulting list. Thus, the items may go through further filtering process. If a hash address found in the source tree does not have its matched hash address in the corresponding target tree, all of the linked source items for the hash address will be discarded. On the other hand, if a hash address found in target tree does not have its corresponding hash address in the source tree, all the linked target items for the hash address will be

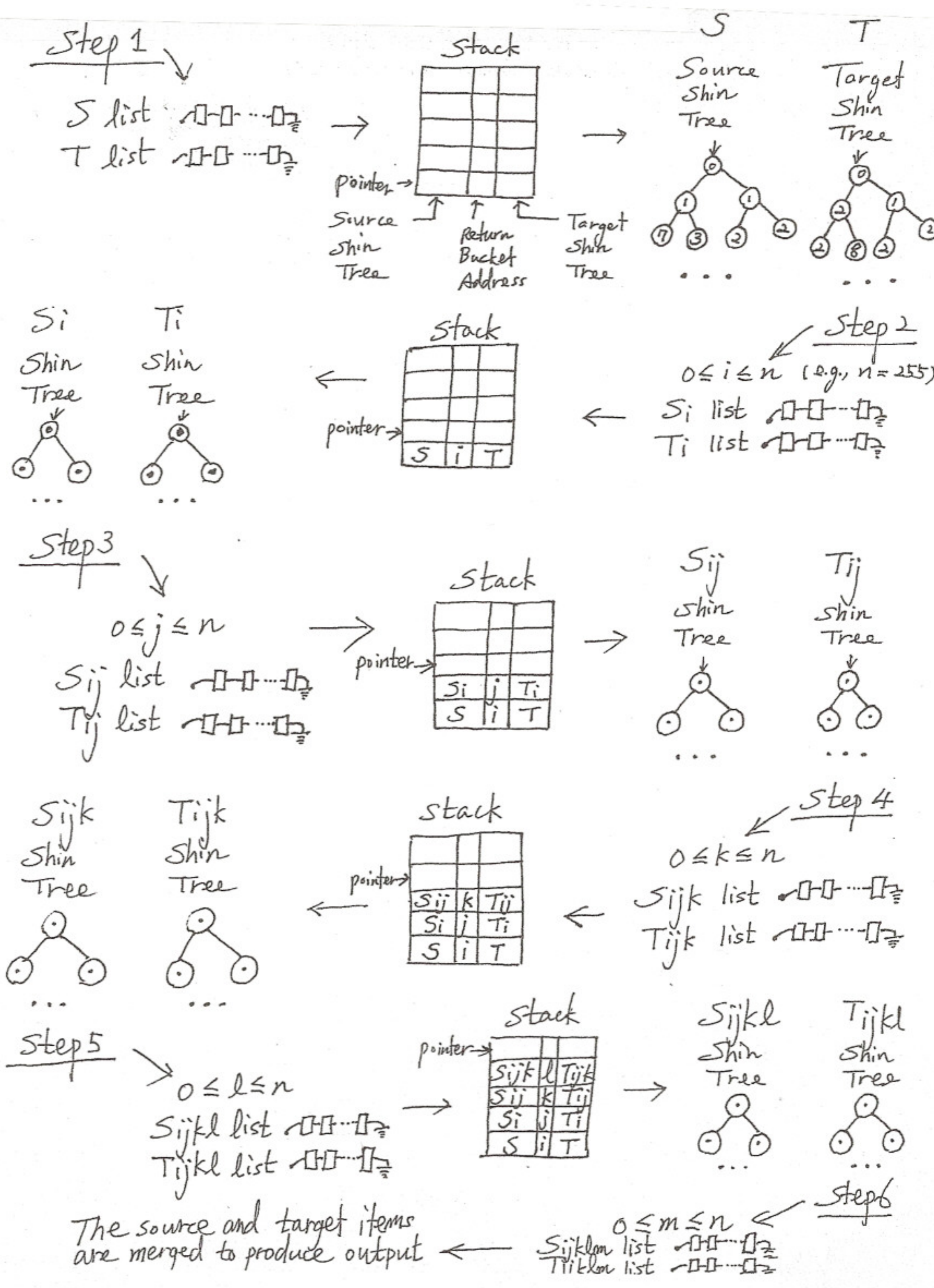

Figure 2: Shin's Algorithm for Massive Cross-Referencing with Shin Sort

discarded. Of course, if a hash address is not found in both source Shin tree and target Shin tree, both traversings will just neglect and skip the process of further work for the hash address. If Shin trees are used in the massive cross-referencing, the above case will be handled more efficiently than when the source and target hash tables are used. All other parts in the filtering process are the same with the original SOFT process and previous description of the algorithm.

Discussion

The process for insertion of an item will be the time complexity that is proportional to one (i.e., O(1)) in Shin tree data structure. One may try binary search tree sort in the author's massive cross-referencing method instead. Then in the first level, the insertion will take O(log(N/B)) time complexity when N is number of input items and B is number of buckets, e.g., 256. While hash tables require scanning time for empty buckets, the BST sort requires time for insertion of which time complexity is not proportional to one. Therefore, if the BST sort is used in the massive cross-referencing method, it's unavoidable to be proportional to NlogN. Before getting involved in filtering any unnecessary data, it seems to have quite a long set-up time.

The author once had a thought about indexed linked list with binary search. It is an alternative way of speeding up the SOFT. It requires O(logB) time complexity for insertion, so the initial set-up will be proportional to NlogB. However, the double linked list in the scheme can make the system too complicated for readers to understand, so indexed linked list was not adapted to suit simplicity.

Considering that Shin sort requires neither scanning empty hash addresses nor insertion process that is beyond O(1) time complexity, one will see that the BST sort cannot be the best choice here. Having source and target hash tables is also cumbersome while Shin tree is offered to save time in scanning empty buckets. Accordingly, this paper shows that the combining the author's algorithm for massive cross-referencing with Shin sort is reasonable and right.

Conclusion

Why is Shin sort useful? It's because it has the fastest searching capability as well as fastest sorting capability, that is, O(1) and O(N) respectively. The author's operation for massive cross-referencing is no exception. The Shin sort provides an efficient way to accelerate the operation. With the Shin sort the massive cross-referencing operation will perform more powerful SOFT touch

on unnecessay data to be filtered. The combination of Shin's massive cross-referencing and Shin sort will guarantee O(N) performance with the best filtering effect. Shin sort and search database systems (S^3 DBMS) will be equipped with the combination to provide the maximum performance. As the author is confident, in every software where sorting and searching is used, Shin sort and search will be employed for its outstanding capability.

Bibliography

[1] Shin, D. K. *A Comparative Study of Hash Functions for a New Hash-Based Relational Join Algorithm.* Pub. #91-23423, Ann Arbor: UMI Dissertation Information Service, 1991.

[2] Shin, D. K. and Meltzer, A. C. "A New Join Algorithm." *ACM SIGMOD RECORD*, Vol. 23, No. 4, Dec. 1994: 13-8.

[3] Shin, DK, "The Theory of Massive Cross-Referencing," *The Proceedings of the Eighth International Conference on Software Engineering and Knowledge Engineering,* Jun. 1996: 545-52.

[4] Shin, DK "A Sorting Method by Dong-Keun Shin," Handwritten Manuscript Displayed in *WWW.DKSHIN.COM,* US Copyright Registration: Txu 857-130, Jul. 1998.

[5] Shin, DK "Character-Based Binary Tree Sorting for Integer Numbers," Handwritten Manuscript Displayed in *WWW.DKSHIN.COM,* US Copyright Registration: Txu 864-961, Jul. 1998.

[6] Shin, DK "A Sorting Method by Dong-Keun Shin," Typed Manuscript Displayed in *WWW.DKSHIN.COM,* US Copyright Registration: TX 4-842-998, Aug. 1998.

Combining Shin's Massive Cross-Referencing (Join) with Shin Sort

Dong-Keun Shin 10/31/2007

Source and Target Name-Key (Hash Address Number)
($0 \le$ Hash Address Number ≤ 255)

Source: KIM (87) KING(123) LION (204) KINO (31) JADE (64) KIN (208) KENT (17) KILE (196) QUEEN (123)

Target: SHIN (251) SMITH (157) MOHAMAD (172) KIM (87) WANG (196) BROWN (22) LEE (138) KING (123) SMITH (~~251~~ 157)

Source Shin Tree

Target Shin Tree

The above example shows how the source and target ~~tuples~~ ~~items~~ keys are inserted into their corresponding Shin tree after the keys ~~(last name)~~ are hashed by a hash function. As both trees

show, the terminal nodes of the trees should point to tuple of its own. The above example has only one field, that is key, in each tuple. Two or more than two key may have same hash address such as "KING" and "QUEEN" in the source list. Their hash address is 123. As shown in the Source Shin Tree, both are linked below the terminal node that points to the linked list. The linked list may be sent to another hash coder for further division process.

To make my explanation easier, a figure in my 1996's paper should be illustrated here. The paper, "The Theory of Massive Cross-Referencing," appears in The Proceedings of the Eighth International Conference on Software Engineering and Knowledge Engineering in June 1996, pages from 545 to 552. The Figure 2 of the paper shows full description of Shin's

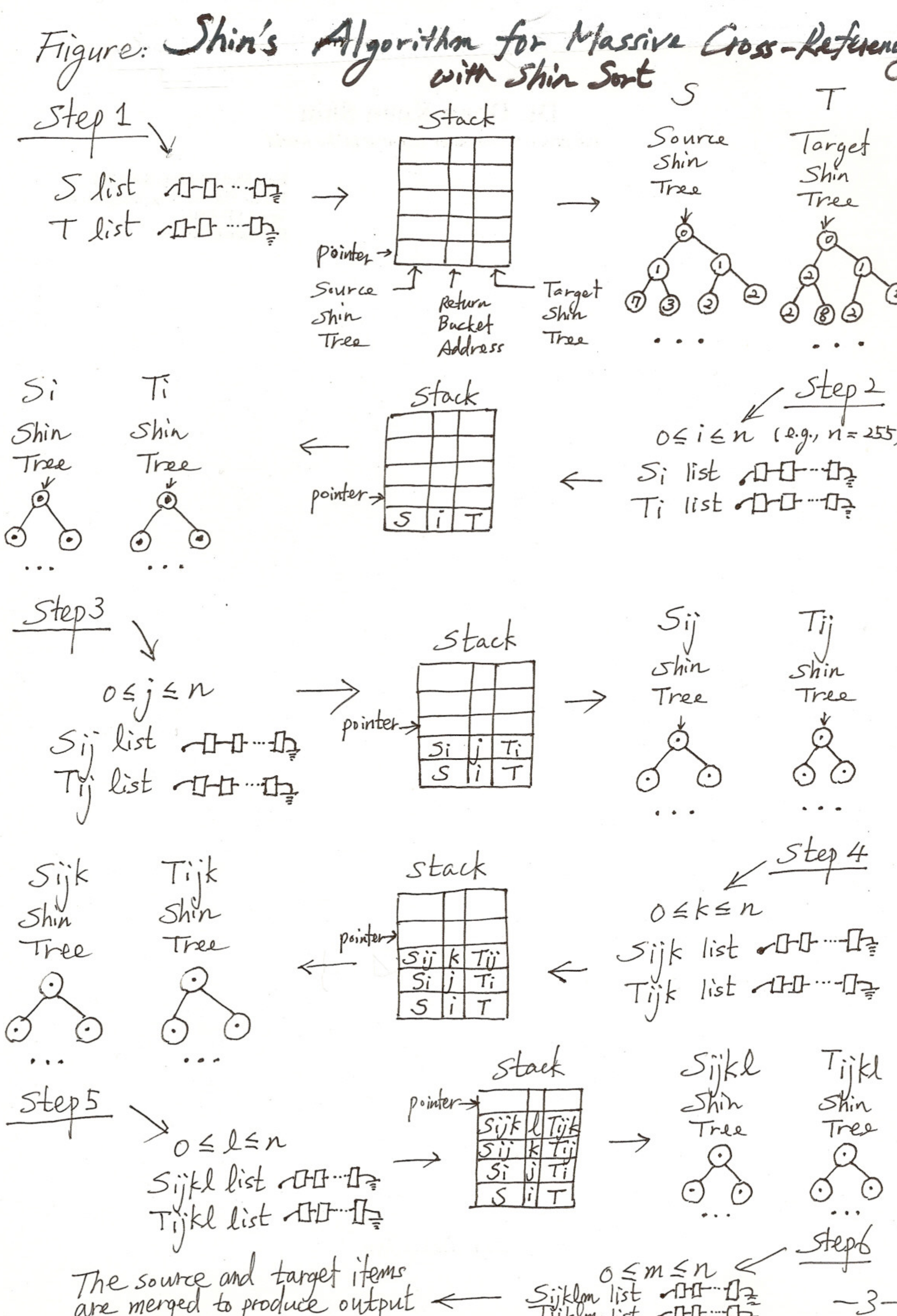
Figure: Shin's Algorithm for Massive Cross-Referencing with Shin Sort
Step 1
S list
T list
Stack
pointer
Source Shin Tree
Return Bucket Address
Target Shin Tree
S
T
Source Shin Tree
Target Shin Tree
Step 2
0 ≤ i ≤ n (e.g., n = 255)
Si list
Ti list
Stack
pointer
S i T
Si
Ti
Shin Tree
Shin Tree
Step3
0 ≤ j ≤ n
Sij list
Tij list
Stack
pointer
Si j Ti
S i T
Sij
Tij
Shin Tree
Shin Tree
Step 4
0 ≤ k ≤ n
Sijk list
Tijk list
stack
pointer
Sij k Tij
Si j Ti
S i T
Sijk
Tijk
Shin Tree
Shin Tree
Step 5
0 ≤ l ≤ n
Sijkl list
Tijkl list
Stack
pointer
Sijk l Tijk
Sij k Tij
Si j Ti
S i T
Sijkl
Tijkl
Shin Tree
Shin Tree
Step6
0 ≤ m ≤ n
Sijklm list
Tijklm list
The source and target items are merged to produce output

Algorithm for Massive Cross-Referencing.

After the Shin sort was discovered by the author, has tables such as S, T, Si, Ti, Sij, Tij, Sijk, Tijk, Sijkl, and Tijkl ~~are~~ may be no longer needed. However, stack is ~~needed~~ needed to keep track of return bucket address in both Source and target Shin tree. Instead of Source hash table and its corresponding target hash table, Source Shin tree and target shin tree can be used to speed up the filtering process. ~~The author once had a thought about having binary tree.~~

All other ideas are the same as before.

While ~~The~~ the source and target lists are being repeatedly divided by a maximum of five functionally different hash coders, ~~until~~ defected unnecessary items are eliminated, ~~and~~ or a group of source and target items are merged if they have an identical

key. The Stack oriented filter technique examines produced hash addresses in both source and target Shin tree, traversing the trees in preorder. Stacked digit numbers of the traversed nodes provide hash address, so the SOFIT compares a hash address provided from the source Shin tree with its closest hash address in the target Shin tree. If both hash addresses are identical, items in the linked list of the Source and target Shin tree have potential to be included in the resulting list. Thus, the items may go through further filtering process. ~~If no matched hash addresses are er~~ If a hash address found in source tree does not have its corresponding hash address in the target tree, all of the items in the linked list in the source tree will be eliminated. On the other hand, If a hash address found in target tree does not have its corresponding hash address

in the source tree, all the items in the source list will be eliminated. It is needless to say that if a hash address is not found in both Source Shin tree and target Shin tree, the traversing will just skip the process of further work. If Shin trees are used in the massive cross-referencing, the above case will be handled more efficiently than when source and target hash tables are used.

All other processes are the same with the process shown in the figure and previous descriptions of the algorithm.

<Discussion>

Insertion of an item will take O(1) time complexity in Shin tree data structure.

If one may try Binary Search tree here instead, the insertion will take $O(\frac{\log(N)}{B})$ for its simplest form. (B: # of buckets, i.e., 256). It's not that bad but it's not the best choice.

Having Source and target hash tables is cumbersome while Shin tree is provided to be used in such massive data operation.

<Conclusion>

Why Shin sort is useful? Because it has the fastest searching capability as well as $O(n)$ sorting capability. Massive Cross-Referencing is not the exception. The

Shin sort provides an efficient way to accelerate the operation. Thanks GOD for all the ideas.

[signature] 10/31/2007