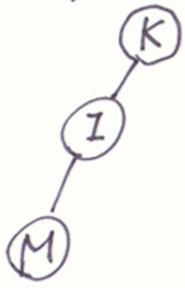
# A Sorting Method by Dong-Keun Shin 7/4/98

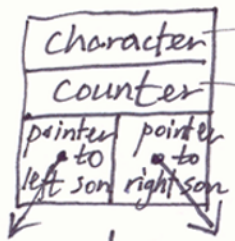
<An Example>

Input: KIM, KING, LION, KIND, JADE, KIN, KENT, KILE, QUEEN

(1) The first input "KIM" is read and ~~the~~ a binary tree is created. 'I' becomes left child (son) of 'K' node. 'M' becomes again a left child (son) of the 'I' node.



The data structure of each node is:

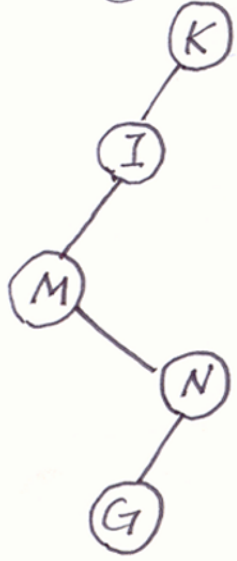


e.g. 'K', 'I', or 'M'

counts number of appearances of an input string in input.

counter should be initialized to be zero. — 7/6/98

(2) Then the second input "KING" is read and inserted to the binary tree. "KI" in the string "KING" is identical to "KI" in "KIM". Thus, 'N' becomes the right child of the node 'M' because the ASCII (or EBCDIC) or internal character value of 'N' is greater than that of the character 'M'.

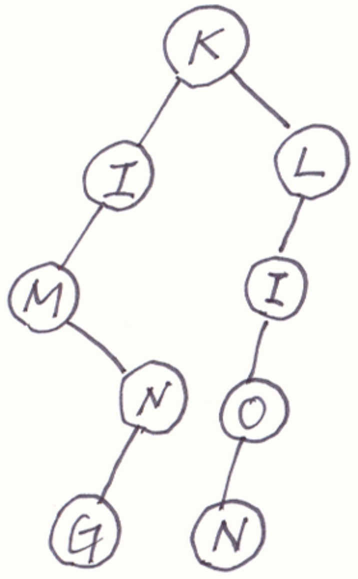


김종열 98.7.4 [illegible]

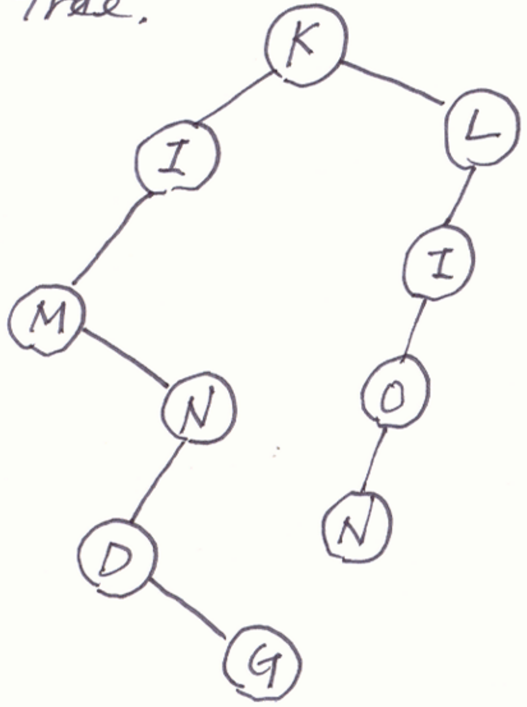
신태근 98.7.4 [illegible]

7/6/98 → Total ~~Eleven~~ Twelve pages 1/12 ~~1/11~~ ~~1/10~~

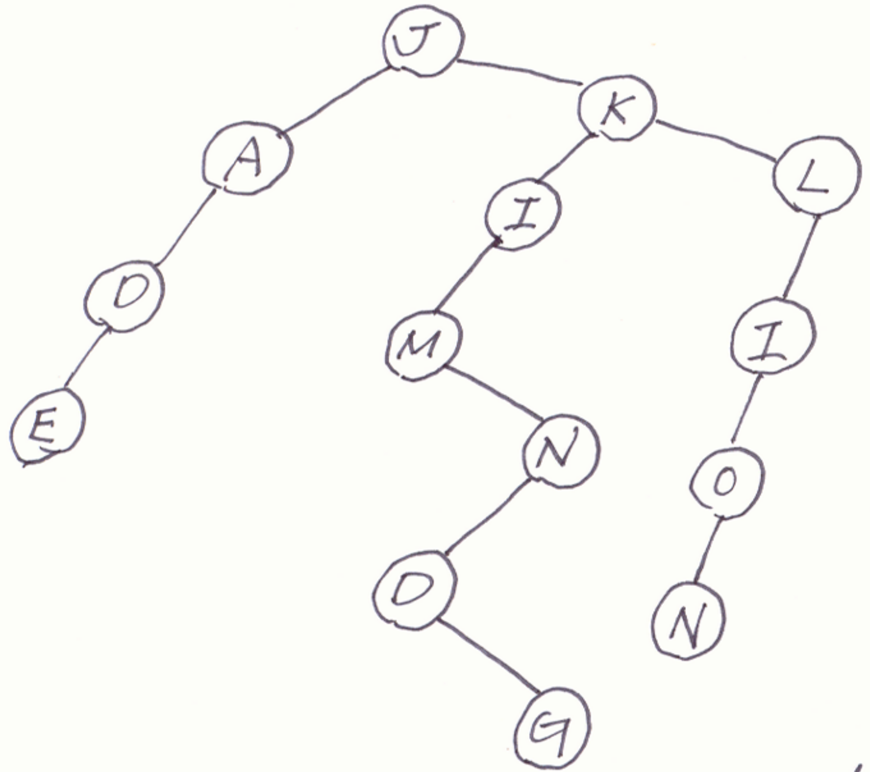
(3) The third input "LION" is read and inserted into the binary tree. The first character 'L' in "LION" is greater than 'K' in root node, so 'L' node will be the right child of 'K' root node.



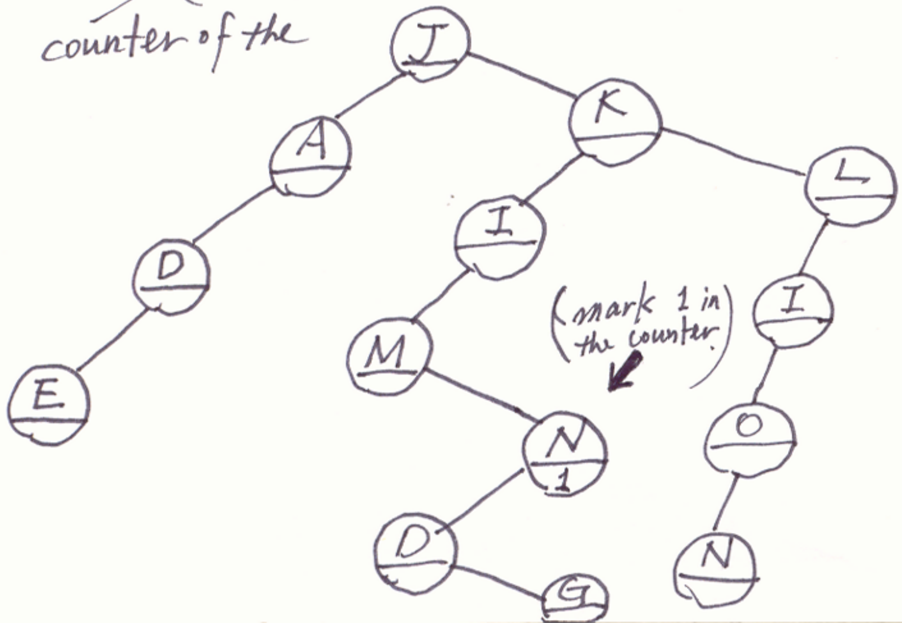
(4) Then next input "KIND" is read and inserted into the tree.



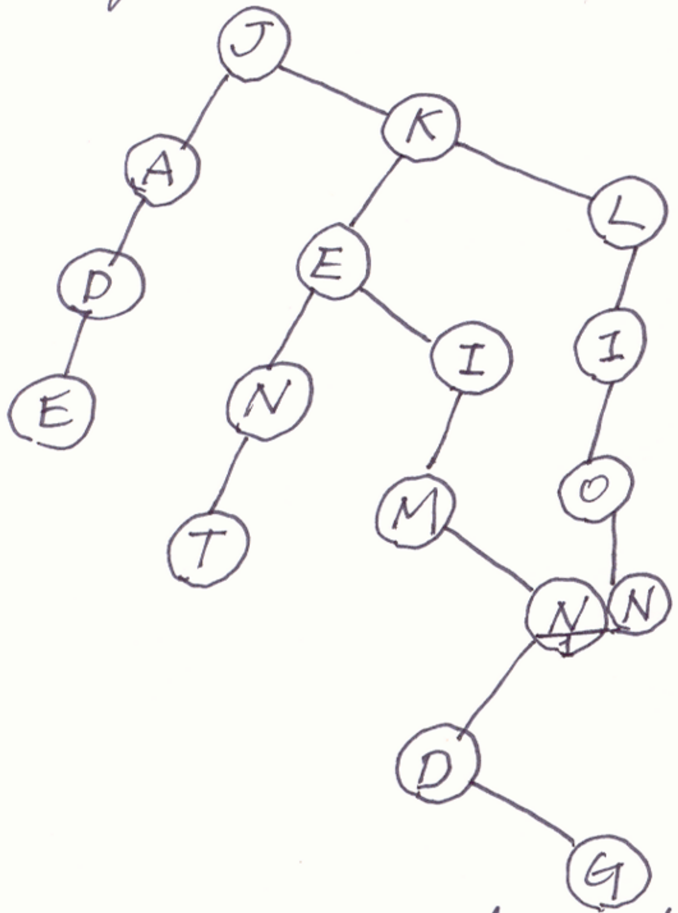
(5) Fifth input "JADE" is read and inserted as shown below. 'J' becomes root node because its internal value is smaller than 'K' in the tree's current root node.



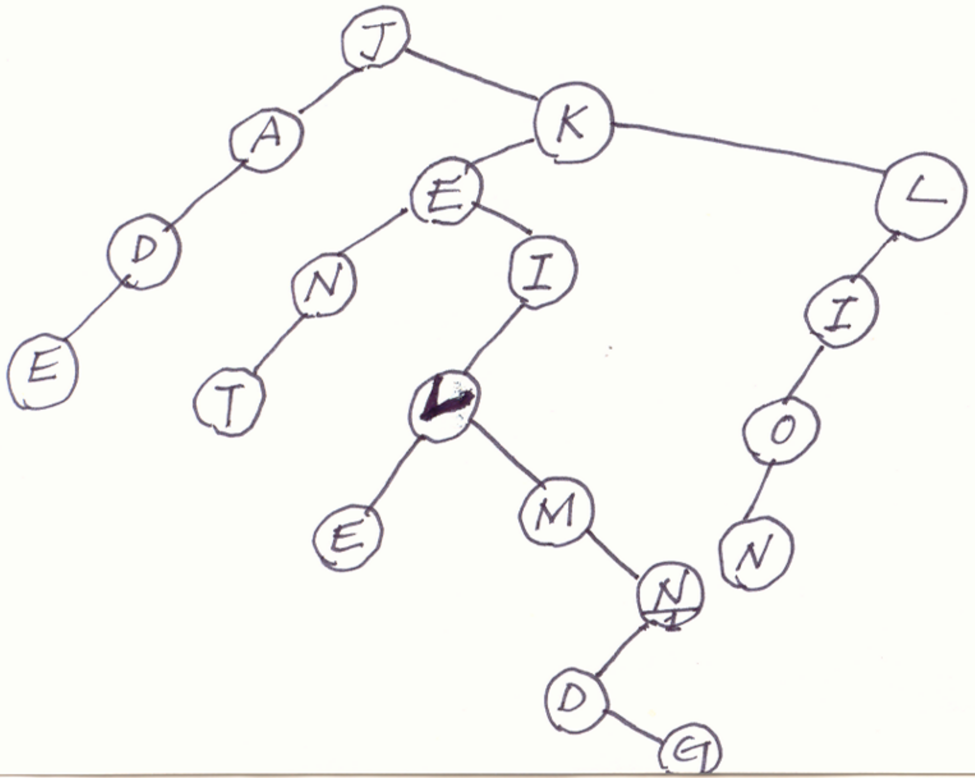
(6) Six input "KIN" is read and inserted. In this case, "KIN" has been already there, so have to mark 1 in the counter of the node 'N' for "KIN"'s first appearance.



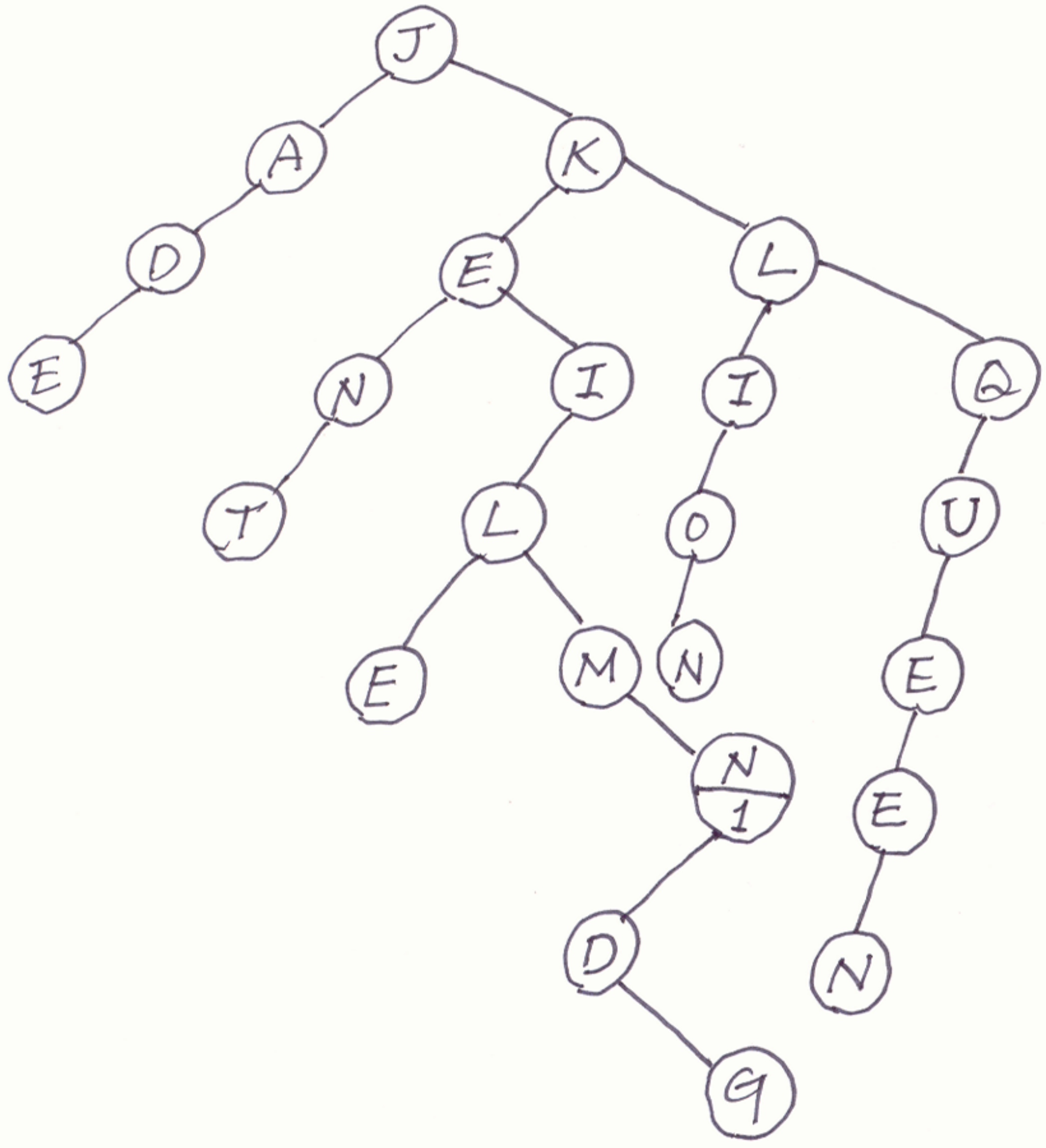
(7) Seventh input "KENT" is read and inserted.
The input string "KENT" is inserted right before "KIM".



(8) Then "KILE" is read and inserted



(9) The last input "QUEEN" is read and inserted into the binary tree. "QUEEN" has the greatest internal value in the tree.



[illegible] 7/4/98.

Sorted list: JADE, KENT, KILE, KIM, KIN
KIND, KING, LION, QUEEN.

양종열 98.7.4 [illegible]
신태균 98.7.4 [illegible]
신승호 98.7.4 [illegible]

<A Sorting Algorithm developed by Dr. Dong-Keun Shin on July 3rd ~ 4th, 1998.>

This sorting algorithm creates ~~a tree~~ a binary tree data structure while reading input ~~data~~ strings. It stores each character in an input string in a node and links them together using each node's pointer for its left child (or left son). However, if a part of the current input string has been already stored in the created tree, it uses the relevant portion of the tree and it uses the lowest node of the portion for different part of the current input string. The lowest node's right child will point to the different part of the string which will create a branch in the tree. This algorithm may read an input string which has been read already, or ~~a portion of an inpu~~ it may read an input string that is identical to a portion of prestored input string in the tree. In both cases, the algorithm increments the counter in the lowest node of the input string or the portion of prestored input string.

To print out strings in the tree, start from the root printing out character by character going to left son only. When a node's left son is empty or nil, the string is complete. Pointers to root ~~or~~ and ~~header~~ nodes for each string should be saved in a stack (7/6/98) for subsequent uses. From a header node's character and its following left nodes's characters ~~and~~ will be printed like "JADE". The printing algorithm for this tree recognizes no more left-out under 'J'. It moves to 'K', then prints out "KENT". The algorithm checks right child of each node from the bottom. It finds 'E' node has its right son. Therefore, 'E' node is skipped, and it goes to 'I' node. After "KI", 'L' and 'E' are left sons followed. Thus "KILE" is printed. The algorithm checks from the leaf node (which is 'E' node) to see the node ~~t~~has right child. If none, it pops up ~~the~~ a stack to move up one level in the tree. It finds 'L' node has right son 'M', and it recognizes "KIM" with no left son in 'M' node. Therefore KIM is printed out.

Because identical keys may be stored or an input key may be the same with a first portion of other key, the algorithm has to check each node's counter (7/6/98)→ while traversing the tree if it

has been incremented. If incremented, the node contains an input's last character; therefore, the string should be printed out. An example "KIN" shows next. Since 'N' node's counter has been incremented to be '1'. It has to print out 'KIN' although
1/6/98 → (one)
the node 'N' has left child 'D' node. Then the algorithm visits 'D' node and finds that the node has no left child, so it prints out "KIND" for next one. The algorithm will check the right child of 'D' node and will find that there is 'G' node. The 'G' node does not have its left child; thus, the algorithm figures out 'G' is at the end of a stored string. The algorithm prints out relevant characters for ~~the~~ stored string in the current path, and the string is "KING". The 'G' node does not have right child; thus, there is no more string left in this brach. The algorithm pops up the stack up to 'K' node and goes to the right child of 'K' node, which is 'L' node. The algorithm saves the current pointer to 'L' node and goes to left childs deep enough to recognize string "LION".

Since "ION" in the string "LION" do not have right child, no other string uses ~~this any part~~ this path. Therefore stack has to be popped again to go back to 'L' node and the algorithm moves to 'Q' node. ~~and~~ It will do the same process it has done for "LION" for the string "QUEEN". It will go down to hit 'N' node in "QUEEN". No more left child in N node will make "QUEEN" to be printed out. Then the algorithm will check if nodes for "QUEEN" have any right child. When it finally figures out the highest node 'Q' does not have right child, the printing strings for output is finished. This printing process is (almost) the same with preorder binary tree traversal.

## <Discussion and Analysis>

This sorting method is an $O(N)$ Sorting algorithm. Comparing with Radix Sorting method which is also $O(N)$ sort method, it is better in several respects:

① Requires much less memory space

② Better in parallel processing.

박 종현 98.7.4 [illegible]
신 태근 98.7.4 [illegible]
82-2-562-1991

## < Conclusion >

Although author does not know whether this algorithm has been discovered by someone already, 7/6/98 ~~he believes that~~ he has not yet seen this sorting method. Dong-Keun Shin, thus, claims this method is a new sorting algorithm. If no one has found this algorithm before, ~~he Shin wants~~ author (Dong-Keun Shin) wants to name this algorithm "Shin Sort" or "Shin's sort"
While considering most sorting algorithms require $O(N \log N)$ or $O(N^2)$, this algorithm require $O(C*K*N)$ ← 7/6/98 ~~$O(K*N)$~~ when K is number of characters in keys; thus, it requires only $O(N)$. Which is good. Although Radix Sort (currently one $O(N)$ method) requires $O(N)$ time complexity, the aforementioned method is better in memory space saving and parallel processing. Hence it is recommendable.

Written by Dong-Keun Shin on July 4, 1998.

< Additional Remarks >   written by Dong-Keun Shin on July 6, 1998

One reason which the new sorting method is better in parallel processing ~~is~~ compared to the Radix method is that the Shin's Sort algorithm does not require heavy data movements as the Radix algorithm does. This bright aspect of the algorithm provides simplicity in designing hardware architecture and software scheme for the author's sorting algorithm's implementation. Combining subtrees together after parallel sorting and creating subtrees in auxiliary storage is relatively easy to implement. This character still remains in main memory space. However, the Radix method is not very recommendable for parallel processing since it lacks inherent characteristic of parallel processing.

One drawback in this new sorting algorithm is that maximum ~~st~~ search for each character in a key string can require $2^7$ (e.g., ASCII characters = 128). Therefore, Upper bound time complexity ~~should~~ of the algorithm should be multiplied by C (e.g. $2^7$ or $2^8$ EBCDIC). O(C*K*N) time complexity is actually O(N) because C and K are constants. (some other constants also may be included, but they are negligible by the same way.) One way to speed up searching character node is using index tables for characters. (which is arrays of pointers for character nodes).

This index table cannot be used everywhere in a tree because of its heavy memory space requirement but it may be used in ~~high~~ the first character of each input keys ~~or in some branches of a tree into which excessively many input keys are inserted. For the second case, a space for counters should be provided in the index table's character elements.~~

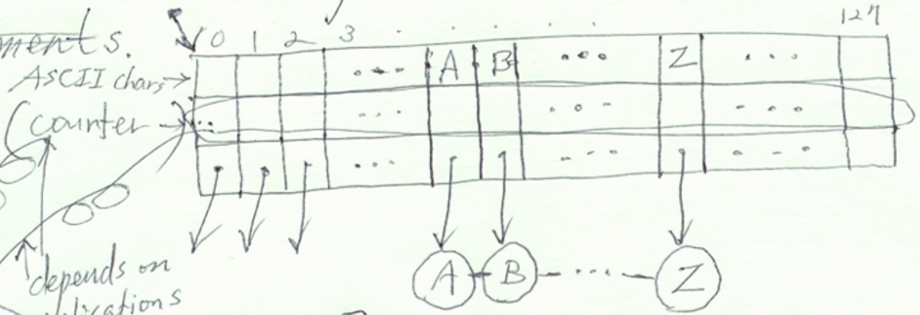


Figure ① Index Table for first character in input keys.

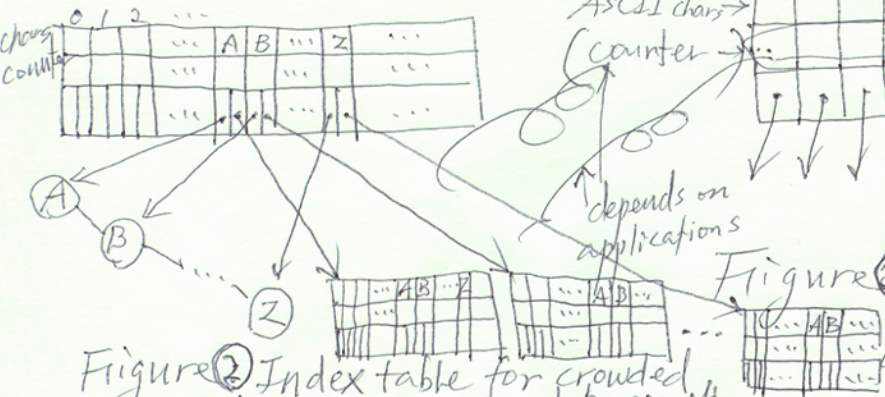


Figure ② Index table for crowded branch

~~Counters in the index table are just temporary based deviation. It will not improve overall time complexity which is still O(N).~~ Figure ① and Figure ②

Hardware architecture and Software architecture will ~~be be provi~~ for this sorting algorithm will be provided in future papers by the author and relevant simulation program for this algorithm will also be written and tested by the author or his colleague.

[illegible], July 6, 1998