Radix Search- an Alternative to Linear Search

Rajesh Ramachandran
Dept. of Computer Science
Naipunnya Institute of Management and Information Technology
Pongam, Kerala
ryanrajesh@hotmail.com

ABSTRACT: *Radix Search is a searching algorithm for finding a particular value in a list by checking each one of its elements. It searches an item by processing individual digits and finally it undergoes a linear method also to make sure the presence or absence of the item in the list. The time complexity of Radix search is* $2^{(n+1)}$ *th of Linear Search where 'n' is the maximum number of digits of a number in the list.*

Keywords: Search Algorithms, Radix Search, Linear Search, Web Search

Received: 12 April 2011, Revised 13 June 2011, Accepted 20 June 2011

1. Introduction

In computer science, a search algorithm is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or elements of a search space defined by a mathematical formula or procedure, such as the roots of an equation with integer variables; or a combination of the two, such as the Hamiltonian circuits of a graph.

Algorithms for searching in explicitly stored databases include the simple linear search, and many other algorithms that use a variety of search data structures, such as binary search trees, heaps and hash tables, to speed up multiple queries over the same database.

There are also many algorithms designed specifically for retrieval in very large databases such as bank account records, electronic documents, product catalogs, fingerprint and image databases, and so on.

2. Comparison between Linear and Binary search

Time complexity of linear search is O(n) and Binary search is O(log n). Binary search runs in O(log n) time whereas linear search runs in O(n) times thus binary search has better performance

But to perform binary search the list must be in sorted order. This pre process is not required for linear search.

The performance of linear search improves if the desired value is more likely to be near the beginning of the list than to its end. Therefore, if some values are much more likely to be searched than others, it is desirable to place them at the beginning of the list.

In particular, when the list items are arranged in order of decreasing probability, and these probabilities are geometrically

distributed, the cost of linear search is only O(1). If the table size 'n' is large enough, linear search will be faster than binary search, whose cost is O(log n)

Binary search can interact poorly with the memory hierarchy (i.e. caching), because of its random-access nature. For in-memory searching, if the span to be searched is small, a linear search may have superior performance simply because it exhibits better locality of reference.

3. Radix Search

Radix search uses the above mentioned advantages of linear search and its performance is better than that of Linear search.

In Radix search, the number will be analyzed by its place value and identified either as an odd or an even number. The odd number would be marked on the right subtree while the even number on the left. Repeat the same for its left and right children item on its place value digit. After all the digits have been analyzed and moved to either right or left subtree it check the item as such whether it is divisible by 3 or not. If the number is divisible by 3 the number moves to its left subtree else it moves to right subtree. Finally, a sequential search is applied to verify whether the item is present in the list.

A tree must be constructed on the same fashion as Radix search. That is based on the place value, the item moves to either left sub tree or right sub tree from the root. And finally the item will be inserted at the beginning of that list. So for a two digit number there will be 4 such list. One with odd odd list, second with odd even list, third with even odd list and the last with even even list and after the divisional check number of list will be 8. That is four list numbers will be divisible by 3 and the other four list numbers will not be divisible by 3.And for a three digit number there will be sixteen such list and for a 4 digit number total list will be 32.That is the number of list will be $2^{(n+1)}$ where 'n' is the number of digits

For example

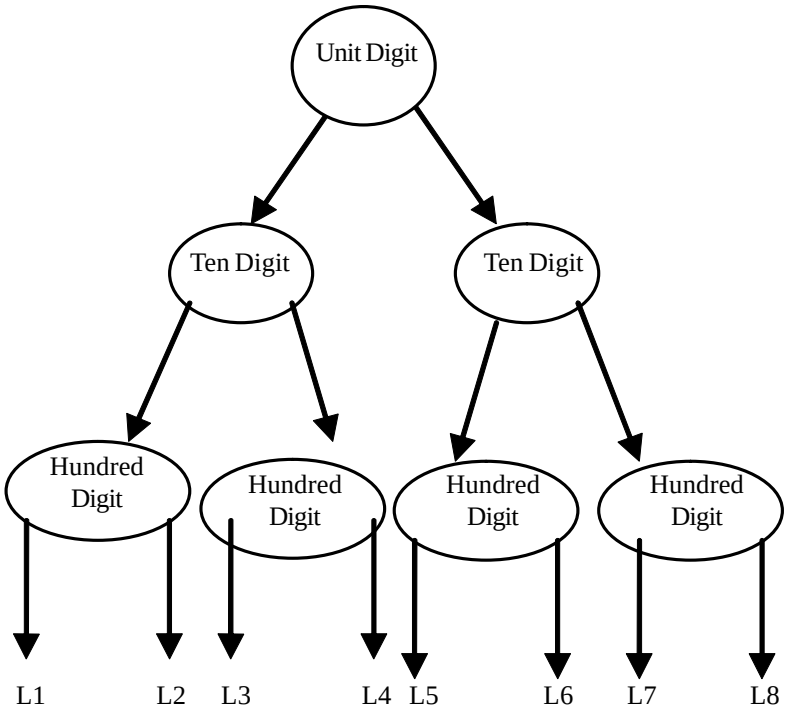

Figure 1.

In Figure 1, all the numbers in the List 1 will be even numbers. In List 2 unit digit and ten digit numbers will be even and hundred digit will be odd. And in List 8 all digits will be odd.List 7 will have numbers with unit digit and ten digit with odd values and hundred digit with even value. Similarly List 3 will be having numbers with Even Odd Even values, List 4 with Odd Odd Even, List 5 with Even Even Odd values and List 6 with Odd Even Odd values

After Comparing all the digits odd or even, compare the item as a whole whether that is divisible by 3 or not. So here List 1 again

will be divided into to two sub list, one with numbers divisible by 3 and the other with numbers not divisible by 3.

Consider the following initial Radix tree with 32 two digit numbers

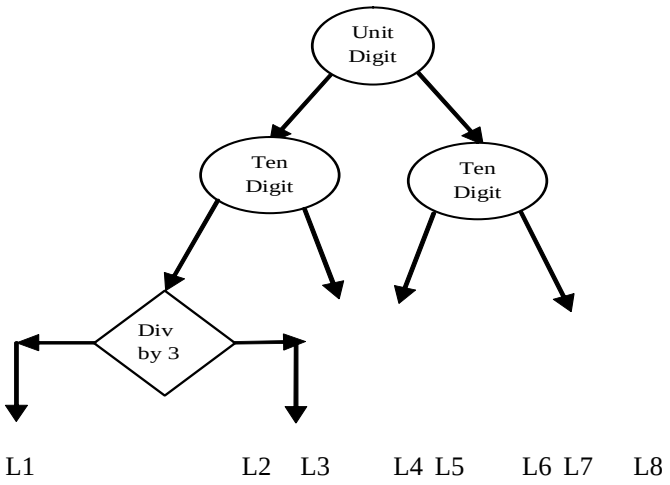

Figure 2.

There will be 8 lists, ie L1,L2,…L7,L8.

List 1 will contain numbers of EVEN EVEN and divisible by 3

List 2 will contain numbers of EVEN EVEN and not divisible by 3

List 3 will contain numbers of EVEN ODD and divisible by 3

List 4 will contain numbers of EVEN ODD and not divisible by 3

List 5 will contain numbers of ODD EVEN and divisible by 3

List 6 will contain numbers of ODD EVEN and not divisible by 3

List 7 will contain numbers of ODD ODD and divisible by 3

List 8 will contain numbers of ODD ODD and not divisible by 3

Let initial radix tree consists with the following numbers in the respective list

L1: 66,24,42,06

L2 :26,88,44,46

L3: 72,18,36,12

L4: 32,56,38,94

L5: 45,69,81

L6:23,49,61,89,85

L7: 15,33,51

L8: 91,13,35,73,77

Suppose if we want to add 75 to the radix tree, first we compare 5, since it is an odd number it goes to right subtree then we compare 7, since this is also an odd number it also goes to right subtree. Now we check 75 is divisible by 3.Since it is divisible by 3 , 75 will be added to the list seven. So the new list7 will be 75,15,33,51

Similarly if we add 52,

The 52 number will be moved to left subtree and then right subtree. The number 52 is not divisible by 3 so this number will be added to List 4.

After inserting 52 the new List 4 will be with the numbers 52,32,56,38,94 the tree will be :

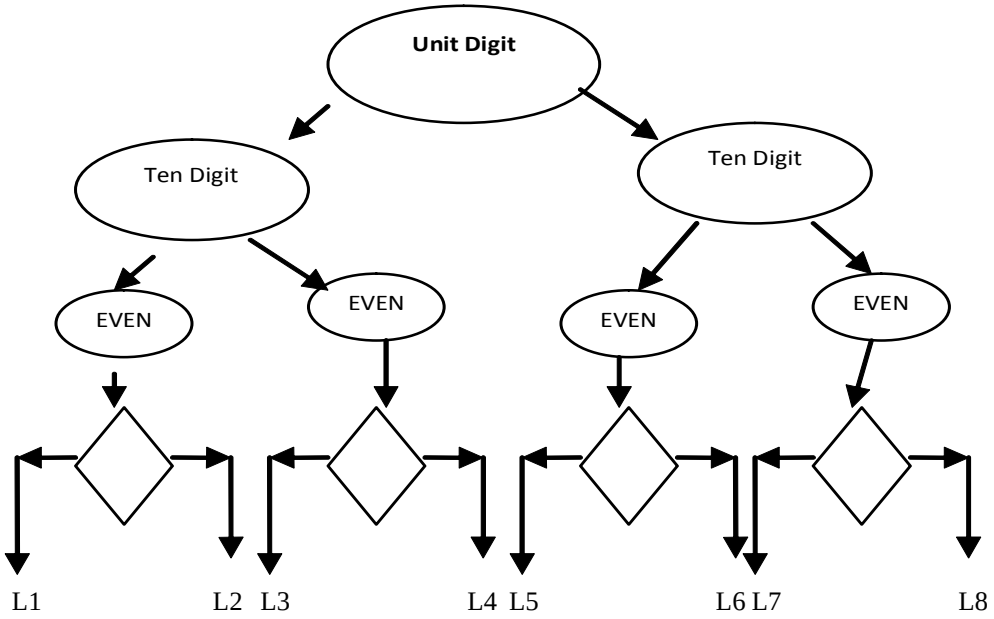

Figure 3. Radix Search Tree

3.1 Radix Search Algorithm

Step 1 Initialize current = root node
Step 2 Read X // where X is the item to be searched.
Step 3. While(X > 0)
Step 4 Get the unit digit of X, Say N
Step 5 Check whether N is ODD or EVEN
If ODD current = right subtree of current
Else current= left subtree of current
Step 6 X=X/10
Step 7 End of While Loop
Step 8 If X is divisible by 3 then move to left subtree of current node
Else move to right subtree of current node
Step 9 Do Linear search for the list which is pointed by the current pointer.
Step 10 End of Radix Search.

For Example

Let X be 89. i.e search item is 89

Here first we take the unit digit. In this case it is 9. since 9 is odd we move to the right subtree of root. Then we read the ten digit, i.e 8. Since 8 is Even we move to left subtree.

Now we check the divisibility of 3. Since 89 is not divisible by 3 we move to right subtree. That is the list L6.

Here the numbers are 23,49,61,89,85

Now we search the item 89 using the Linear search method.

3.2 Time Complexity

In Radix search the initial comparisons will reduce the entire list based on the maximum number of digits of a number in the list.For a two digit number list would be reduced to one eighth and for a three digit number list would be reduced to one sixteenth and so on. That is the list would be reduced to $2^{(n+1)}$ where 'n' is the number of digits.

For example for a two digit number , list could have a maximum of 100 numbers and after the initial two digit comparison the search list will be reduced to 25 and after performing the divisional check it will reduce again to 13. So the time complexity of Radix search is $2^{(n+1)}$ th of Linear Search where 'n' is the number of digits. So the time complexity of Radix Search is better than Linear Search. The time complexity of Radix search is not better than binary search but, the initial list need not be in sorted order in Radix search like linear search.

4. Conclusion

Radix search could be used as an alternative for linear search. As mentioned earlier linear search have some advantages over binary search. These advantages are also applicable to Radix search. Radix search could be well implemented if the list is represented as linked list. The number of comparison could be minimized by placing the frequently referred item in the front of the list.

References

[1] http://en.wikipedia.org/wiki/ Sequential_search

[2] http://en.wikipedia.org/wiki/ Binary_search_algorithm

[3] Aaron M Tanenbaum, Moshe J Augenstein, (1986). Data Structures using C, Prentice Hall International Inc., Emglewood Cliffs, NJ.

[4] Robert L Cruse, (1999). Data Structure and Program Design, Prentice Hall India 3rd ed.

[5] Robert Kruse, C L Tondo, Bruse Leung, (2002). Data Structures and Program design in C, Pearson Education, 2nd Ed.

[6] Alfred V Aho, John E Hopcroft, Jeffrey D Ullman, (2003). The Design and Analysis of Computer Alogorithms, Pearson Education.

[7] Alfred V Aho, John E Hopcroft, Jeffrey D Ullman, (2006). Data Structures and Algorithms, Pearson Education,2nd Ed.

[8] Sara Baase, Allen Van Gelder, (2006). Computer Algorithms Introduction to Design and Analysis, Pearson Education, 3rd Ed.

[9] Seymour Lipschutz, GAV Pai , (2007). Data Structures, Tata McGraw Hill

[10] Robert Lafore, (2007). Data Structures and Algorithms in Java, Waite Group Inc.

[11] Rajesh Ramachandran, (2010). Kakkot Search- A New Searching Method, National Conference on Software Engineering NCSOFT 10, Cochin University of Science and Technology on May 24-25, ISBN 978-83-80095-09-7, pp211.