www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 4 Issue 12 Dec 2015, Page No. 15397-15402

Implementing Radix Sort With Linked Buckets Using Lsd & Msd And Their Comparitive Analysis And Discussion On Applications

Mr. Murali Krishna Senapaty [a] , Mrs. Padmaja Patel [b], Mr. Ranjeet Panigrahi[c]

[a , b, c] Dept. of CSE, Gandhi Institute of Engineering & Technology, Gunupur, Rayagada, 765022 (India)

[a] muralisenapaty@gmail.com
[b] amar.padmaja@gmail.com
[c] ranjeetpanigrahi@gmail.com

Abstract: Here I have presented about implementing radix sort with linked buckets concept to reduce the memory usage for Large Data Set. In this research paper I have discussed about the various ways to implement radix sort, problems with the radix sort, brief study of previous works of radix sort & elaborating the use of radix sort for large data set. I try to analyse the memory usage problems of radix sort through this algorithm. Here I have taken the help of C language to execute and analyse the algorithm.

Introduction:

Radix sort is a method that is commonly used to sorts data with integer keys. But it is not limited to integers; the radix sort can be useful for sorting text, binary and floating values also. The Radix sort can be mainly categorised in two ways i.e: either by processing digits of each number starting from the least significant digit(LSD) or by processing the most significant digit (MSD).
Here we are focussing more on LSD radix sort where the process the integer representations starting from the least digit and move towards the most significant digit.
But the MSD radix sort is suitable for sorting elements in lexicographical order. The elements can be strings and fixed length integers.[4][7]

When radix sort is used for sorting integer values, it sorts *n* keys based on their individual digits (within 0 to 9). So, the worst case time complexity of Radix sort can be measured as *O(s.n)* for n keys and the integers of word size *s*.[1]

Discussion on the application areas:

In many application areas the need of efficient sorting routines are required. And basis of their own efficiency it terms of time complexity, amount of memory usage during sorting they are used in different application areas. In the database systems there is an extensive use of sorting operations.[4]
The digital computer keeps all of the data in electronic binary numbers, so for that it processes the digits of integer representations by grouping of binary representations.
Therefore it is important to view the efficient sorting routines any programming platform. Also with the evolution of latest computer architectures there is a huge need to explore efficient sorting techniques on them.

2. Related Works:

Generally Radix sort is suitable on computers having a large memory space, parallel processors but this method is quite fast and stable method when there is a huge amount of dataset is to be sorted as its time complexity purely does not depend on the number of key values N. Radix sort is classified by Knuth as "sorting by distribution". When the key values of data set have equal length then the sorting is much faster compare to unequal length keys. The unequal length keys increases the no. of passes which increases time complexity. [4]
Radix sort is used by vector multiprocessors on large set of data for executing faster than a highly optimised library sorting. So on a dual processor the performance of radix sort is better than all other comparison based sorting. [5][6]. An optimization on the parallel radix sort algorithm

reduces the time complexity and maintains balanced loads on all the processors.[13]

3. Discussion of Applications :

The LSD radix sort is not suitable for sorting the strings as it is starting with list significant digit. The MSD radix sort is useful as it splits the strings into groups according to their 1st character and then arrange the groups in ascending order. Then by applying the algorithm recursively on each group by ignoring the 1st character of each string the group can be sub divided into sub groups. Due to it the searching for strings as per dictionary order can also be faster.[8]

The Drawback of radix sort is if it implemented in string and if pure English words are sorted then also minimum 26 buckets are needed(each sublist or bucket represents a character). If the string contains alphanumeric words then the no. of sublists needed even more.[9][13]

4. LSD(Least Significant Digit) Radix Sort:

The LSD Radix sort Execution steps while implementing on integers:

- Find the count of number of digits in the largest key in the list
- Now repeatedly execute for each digit of keys from LSD to towards MSD
    - Now read significant digit of each key and put the key into the respective bucket [digit]
    - Recollect the keys in same order and make a list
- The list will be sorted and stable.

Here the number of buckets is according to the digits starting from 0 to 9. The size of bucket can be:
1. Fixed length 2. Variable length.

In fixed length buckets the need of memory is large. The memory usage will be as:

- The size of fixed length bucket has to be according to the number of keys in the list. If the number of key elements are N, then a bucket size will be N and total size required for all buckets for integer sorting is 10*N.

In focus on effective memory utilization the buckets cane be of variable length. It is possible in two ways:

1. The bucket can be allocated dynamically and instantly according to the number of times that each digit occurs in the keys of array while retrieving digits in the order from LSD to MSD. The buckets can implement with array concept.
2. The buckets can be created with the help of an array of pointer size 10 where each pointer represents a specific digit bucket linked list. Now while retrieving a digit from a key, the key can be stored by creating a node and linked to the respective digit bucket.

Drawback: When a small lists of keys present for sorting, then implementing LSD Radix sort proves inefficient in terms of time complexity.[3]

4.1 : LSD Radix Sort with Fixed Length Bucket:

It is a standard way for radix sort where the buckets generally a two dimensional array where the row index identifies the bucket number and each bucket allows to store keys into them as per the digit. [5] So, here the memory of bucket size is fixed which are not properly used. So when a large list of key values present it needs each bucket of large size.
Algorithm:

Step 1: allocate 10 no. of buckets having size as per no. of keys K to sort
Step 2: L = largest element of the keys
Step 3: N=total no. of digits in the L
[Pass represents position from LSD(least significant digit) to MSD(Most significant digit)]
Step 4: for Pass =1 to N
Step 4.1: initialise all the buckets
Step 4.2: for i=1 to K
get the digit from ith key from position Pass and then Put the ith key into the bucket[digit]
[end of for-4.2]
Step 4.3: recollect the keys in orderly from buckets (0 to 9) and store into Array.
[end of for – 4]
Step 5: stop

4.2: Radix Sort with Bucket allocated dynamic array:

In this method at first each digit is obtained from LSD towards MSD. Then by finding the number of occurrences of each digit the respective bucket size can be allocated. Then according to the digit obtained the keys are stored into the buckets.

Consider a list of keys:

180, 045, 075, 080, 002, 044, 802, 066

In the first pass the least significant digit of each key, produces an array of bucket sizes as follows:

Digit	**Key elements to store**	**No. of keys to store in respective bucket**	**Dynamic Array Created**
0	180, 080	2	Created Bucket0[2]
1	no key values	0	No bucket is created
2	002, 802	2	Created Bucket2[2]
3	no key values	0	No bucket is created
4	044	1	Created Bucket3[1]
5	045, 075	2	Created Bucket5[2]
6	066	1	Created Bucket6[1]
7	no key values	0	No bucket is created
8	no key values	0	No bucket is created
9	no key values	0	No bucket is created

Then the after storing based on digit obtained the keys are recollected back to the array. Then during 2nd pass the digit previous to LSD is obtained and the same process is followed to allocate memory for buckets.

Algorithm:

Step1: L = largest element of the keys in the array list

Step 3: N=total no. of digits in the L

[Pass represents position of digit from LSD(least significant digit) to MSD(Most significant digit)]

Step 4: for Pass =1 to N

Step 4.1: Find the sizes of each bucket based on the count of number of occurrences of each digit (0 to 9)

Step 4.2: initialise all the buckets from 0 to 9 according to the respective bucket sizes

Step 4.3: for i=1 to K

get the digit from ith key from position Pass and then Put the ith key into the bucket[digit]

[end of for-4.2]

Step 4.3: recollect the keys in orderly from buckets (0 to 9) and store into Array.

[end of for – 4]

Step 5: stop

4.3: Radix Sort with Buckets using Linked List:

Algorithm:

Main procedure:

Step 1: Create an array of pointer of node type

Step 2: Create array A[n], I, max, c

Step 3: Store a smallest number in max and set c=0

Step 4: Find the largest key in the Array A[n] and store in max

Step 5: c= Count of number of digits in max

Step 6: call LinkedBucketSort(A[n], c)

Step 7: Display the sorted list of keys from array A[n]

Step 8: stop

Procedure LinkedBucketSort(A[n], N)

Step 1: Set unit position N1=0

Step 2: Repeated execute for each N1 less than N

Step 2.1: Set the Bucket pointers to NULL

Step 2.2: for (i=1 to nth index of Array)

Step 2.2.1: digit=Extract digit from A[i] as per N1

Step 2.2.2: Create a new node and store the Key A[i]

Step 2.2.3: if bucket pointer P[digit] is NULL then store node address

Otherwise Insert the node at the end of Linked list P[digit]

[end of for-2.2]

Step 2.3: Recollect the node and store their Key into Array A

[End of loop – 2]

Step 3:stop

'C' Program Implementation:

```
//Implementation using C Program for LSD radix sort using linked buckets
#include<stdio.h>
#include<conio.h>
```

```
#include<math.h>
#include<alloc.h>
struct node{ int info; struct node *link;};
struct node *p[10];      //an array of node pointers
void linkedbucketsort(int [],int);   //function for sorting
void main()
{
int a[10], i,max= -32767,c=0;
printf("\nenter 10 numbers");
for(i=0;i<=9;i++)        {        scanf("%d",&a[i]);
if(a[i]>max) max=a[i]; }
for(;max>0;max=max/10){c++;}
//max is containing the largest number
//variable c is used to contain count of digits in max
linkedbucketsort(a,c);
printf("\nafter sorting:");
for(i=0;i<=9;i++) printf(" %d",a[i]);
getch();
}
void linkedbucketsort(int a[10],int n)
{
        int digit,i,j,n1;
        struct node *fresh,*ptr;

    n1=0;      //n1 initially containing unit position i.e: 0
   while(n1<n)
   {
        for(i=0;i<=9;i++)p[i]=NULL;//initializing the bucket pointers to NULL

        for(i=0;i<=9;i++)
        {
                digit=int((a[i]/pow(10,n1)))%(10);
        //to extract the digit as per the position n1

                fresh=(struct                node *)malloc(sizeof(struct node));
                fresh->info=a[i];
                fresh->link=NULL;

                //Linking the node at the end of respective Linked bucket P[digit]
                if(p[digit]==NULL)
                        p[digit]=fresh;
                else
                {
                  for(ptr=p[digit];ptr->link!=NULL; ptr=ptr->link);
                  ptr->link=fresh;
                }
        }

        j=0;

        //recollecting the keys from linked buckets into array in order.
        for(i=0;i<=9;i++)
        {
                for(ptr=p[i];ptr!=NULL;    ptr=ptr->link)
                {        a[j]=ptr->info;  j++; }
        }

   n1++;
   }

}
```

5. MSD Radix Sort:

The MSD (most-significant-digit) Radix Sort checks the digits in the keys in a left-to-right order, working with the most significant digit first and moves towards LSD. Here the key values are divided into P ordered partitions with the help of buckets using MSD of key values. Then each partition is recursively sorted successively. [3][13]

The MSD radix sort works as follows in recursive approach as:

1. Read the MSD of each key from the list.
2. Store the keys into the buckets according to the digit obtained. Store all the elements with the same digit into one bucket.
3. Then recursively sort each bucket, based on the next digit to the right side onwards.
4. Finally join the buckets together to get sorted list.

The Recursive approach makes execution is slower so there is another modified approach to implement MSD in more simple way:
Advantage: When a huge amount of dataset is present then this radix sort can be useful.

5.1: MSD(Most Significant Digit) Radix Sort with insertion sorting concept:

The performance of radix sorting method is very poor when there is a small number of key values

exist for sorting. Again if the number of digits in each key is more that means the key elements are big numbers then it increases the number of passes which in turn increases time complexity unnecessarily. [15] So in this instance after one phase of radix sort an insertion sort can be applied on key pointers. It in turn gives an attractive reduction of time complexity.[3][10]

Steps to execute Modified MSD(Most Significant Digit) Radix Sort:

1. Read the MSD of each key from the list.
2. Store the keys into the buckets according to the digit obtained.
    a. When multiple key elements obtained digit is same then store them using insertion sort procedure.
3. Finally join the buckets together to get sorted list.

If we go for MSD radix sort then its time complexity can be measured as O(N*K) , where N is the number of Passes and K is the time taken for each pass. But if we implement single pass MSD radix sort followed by insertion sort in each buckets then the time complexity can be O(K+L) where K is the time taken for 1st pass and L is the time measured for insertion sort operations in buckets.

Here as insertion sort executes after radix sort, so it needs a minimum number of changes to make the sorted order of keys. So it increases the efficiency of the algorithm.

5.2: MSD Radix Sort with insertion sorting concept having Buckets using Linked List:

[2][11]Again if we go for variable length buckets using linked list concepts then it increase the efficiency in terms of time and space complexity. Following elaborates using MSD Radix using linked buckets:

Algorithm:

Step 1: Create an array of pointer of node type for buckets and initialize to NULL

Step 3: repeatedly Obtain the MSD(most significant digit) from each Key

Step 3.1: Create a node and store Key into it

Step 3.2: if Bucket[digit] is NULL then insert the key node into it Otherwise Insert the Key node in the respective position to maintain sorted order of keys in bucket.

[end of loop]

Step 3: recollect the keys in orderly from buckets (0 to 9) and store into Array.

Step 4: stop

Efficiency Comparison on various radix sorting algorithms:

Type of radix sort	Efficiency in terms of time and space utilization
LSD Radix sort with fixed size array buckets	➢ The time complexity is N^2 ➢ the amount of memory space needed for buckets is more. ➢ Useful for sorting of integers, characters
LSD Radix sort with dynamic arrays buckets	➢ The time complexity is N^2 ➢ the amount of memory space needed for each bucket <=total no. Of digits. ➢ Useful for sorting of integers, characters
LSD radix sort with linked buckets	➢ The time complexity is N^2 but the amount of memory space needed for each bucket is less. ➢ Space complexity will be affected by each node for storing address. ➢ Useful for sorting of integers, characters
MSD Radix Sort in recursive method	➢ The Recursive process affects the time complexity. ➢ Highly useful for sorting the Strings in a large data set
Modified Single Pass MSD Radix Sort with Insertion Sort	➢ The amount of time required to sort is lesser than $O(N^2)$ ➢ Highly useful for a sorting small amount of key values.

6. Conclusion:

In the paper we have discussed on LSD Radix sort using fixed length buckets, dynamically allocated buckets and linked buckets. Also we have compared the performance of MSD radix sort in different application areas. We have observed that MSD recursive radix sort is best suitable for string sorting in a large data set, where as the Single Pass MSD radix sort is best suited for a small set of key values. Even we have observed the effective use of memory in terms of buckets in algorithms of LSD Radix sort using dynamic allocated buckets.

Further we observed that Radix sort is used much with parallel processors, Vector Processors and Database. So when we sort a large set of data and go for search operations the radix sort is having attractive performance.

References:

1. https://en.wikipedia.org/wiki/Radix_sort
2. Prof.Ramesh chandra pandey, Study and comparison of various sorting algorithms, thapar university, Patiala.
3. The Computer Journal, Vol 30, No.1, 1990, 000, By I.J.DAVIS, Dept of Computing and Physics, Wilfrid Laurier University, Waterloo, Ontario, Canada N2L 3X7
4. Avinash Shukla, Anil Kishore Saxena / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com Vol. 2, Issue 5, September- October 2012, pp.555-560, 'Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set in Distributed Computing Environment'
5. Marco Zagha and Guy E. Blelloch, Radix Sort For Vector Multiprocessors, Computer Science,Carnegie Mellon University,Pittsburgh, PA 15213-3890
6. Ahmed M. Aliyu, Dr. P. B. Zirra, Computer Science Department, Adamawa State University, Mubi, Nigeria, Evaluation of Power Consumption of Modified Bubble, Quick and Radix Sort, Algorithm on the Dual Processor, Ahmed M. Aliyu et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1) , 2014, 956-960
7. A. Avinash Shukla1, B. Anil Kishore Saxena 2, Modified Pure Radix Sort for Large Heterogeneous Data Set Modified Pure Radix Sort for Large Heterogeneous Data Set, IOSR Journal of Computer Engineering (IOSRJCE), ISSN: 2278-0661 Volume 3, Issue 1 (July-Aug. 2012), PP 20-23, www.iosrjournals.org
8. Arne Andersson, lund university and Stefan nilson, Helsinki university of technology, Implementing Radix Sort, The ACM Journal of Experimental Algorithmics. Volume 3, Article 7, 1998.
9. Rohit Joshi, Govind Singh Panwar, Preeti Pathak, Dept. of CSE, GEHU, Dehradun,India, Analysis of Non-Comparison Based Sorting Algorithms: A Review, International Journal of Emerging Research in Management &Technology ISSN: 2278-9359 (Volume-2, Issue-12)
10. https://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf
11. http://codersmaze.com/sorting/bucket-sort/
12. R.Sedgewick, the analysis of Quicksort programs, Acta Informatica 7, 327-355 (1977)
13. Ame, A.A. & Stefan, N.S.Implementing Radix Sort, A.C.M. Journal of Experimental Algorithms, 1998.