International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 5, Issue 3, May-June 2016 **ISSN 2278-6856**

FDM SORT: An External and Distributed Sorting

Chintha SivaKrishnaiah[1*], Puttumbaku.ChittiBabu[2]

[1]Asst. Professor, Annmacharya PG College of Computer Studies, Rajampet, Kadapa(District), AndhraPradesh, India,

[2]Professor ,Annmacharya PG College of Computer Studies, Rajampet, Kadapa(District),Andhra Pradesh, India,

Abstract

Distribution is a competent method for sorting proving by many of them in terms of complexity with little comparisons. In this paper we put forward an approach to sort elements by distribution using Fibonacci Sequence and a small number of arithmetic calculations. It has no comparisons in case of distinctive set of data elements. If elements are repetitive then only comparison exist at the distributed position where repetitive element exists. The sorting technique uses array of four dimensions or sparse matrix or any other possible data structure that hold the distributed element externally. Spectral test prove that the distribution by dimensional approach is a proficient method for unique distributions. This algorithm is capable to sort either set of positive or set of negative values in any range. Due to limits it can not sort a set of values that contain both positive and negative.

Keywords: fibonacci sort, distribution sort, external sort, integer sort

1.Introduction

Sorting is a process of arranging the things according to some class or kind. Sorting by distribution is a class of sorting methods in which sorting operation is carried out by means of distribution based on the constituent components in the elements. It contains sorting without key comparison and having running time complexity is O(n). Some popular distribution sort techniques are Radix Sort, Bucket Sort, Counting Sort, Bead Sort, Pigeonhole Sort, Burst Sort, Proxmap Sort, Flash Sort and American Flag Sort. The famous Fibonacci sequence defined in the thirteenth century by Leonard Fibonacci has numerous applications used extensively in computer science. Originally Fibonacci sequence was proposed to describe a vast diversity of phenomena from nature [1][2]. One of the phenomena was linear order. This linear order of Fibonacci sequence was used by us to distribute the numbers to achieve sorting with zero comparisons in case of unique set of values, one or more comparisons in case of repetitive values in the set.

2.RELATED WORK

Address calculation sort uses a hashing method that is order-preserving, similar to hash sort.it has objective to reduce memory requirements. However, the address calculation has the problem that if the distribution is not uniformly distributed, then the address calculation sort degenerates into an O (N2) time complexity.

Radix sort is one that the digits of each data value are used in the sort. Radix sort for M-sized data element with N elements has a time complexity of O(mn). If the sub data elements become very dense, then M becomes more approximately logN, then the Radix Sort degenerates to a $O(N\log_N)$ algorithm. So hence, the Radix Sort depends on M much less than N by a sizable ratio.

Hash Sort is a general purpose non-comparison based sorting algorithm, which has some features not found in conventional sorting algorithms. The Hash Sort targets to gain algorithmic performance by means of hash function to map the data as a key within the memory. But Hash Sort algorithm [6] has a major problem that the data requires to map into sorting order must exist within some range. Due to this range limitation it has limited scope of application.

Fibonacci numbers are used in a polyphase version of the merge sort algorithm in which an unsorted list is divided into two lists whose lengths correspond to sequential Fibonacci numbers – by dividing the list so that the two parts have lengths in the approximate proportion φ.

3.FDM SORT

FDM stands Fibonacci Divide and Modulus. The term 'Fibonacci' indicates the famous Fibonacci sequence and division, modulus arithmetic operations are key arithmetic components used in the technique. The elements that are usedto map into sorting order are first segregate into ascending order using Fibonacci sequence. sequential characteristic of Fibonacci sequence helps in initial distribution. Later a value is calculated using the sequence and the operations division and modulus produce more distinct distribution mapping. This mapping requires four dimensional external space in which first and second dimensions maps value using Fibonacci order, third and fourth dimensions maps value using relative characteristic of map value with Fibonacci value.

FDMSORT PROCEDURE

A – An array of N unordered elements

OUT – An array of N elements in ascending order

1. Begin
2. Initialize external process data structure P with '∞'
3. For each element of A calculate
   - 3.1. P1 = fib position for absolute value of element A[i]
   - 3.2. P2 = difference between |A[i]| and Fibonacci value of P1
   - 3.3. P3 = P1 / N
   - 3.4. P4 = P1 % N
   - 3.5. P5 = P2 / N
   - 3.6. P6 = P3 % N
   - 3.7. LET (P3, P4, P5, P6) is distributed position of element A[i] in P.
4. Collect elements from P that are not equal to '∞' and put in OUT array.
5. Return OUT array
6. End

Figure 1. Algorithm of FDM Sort

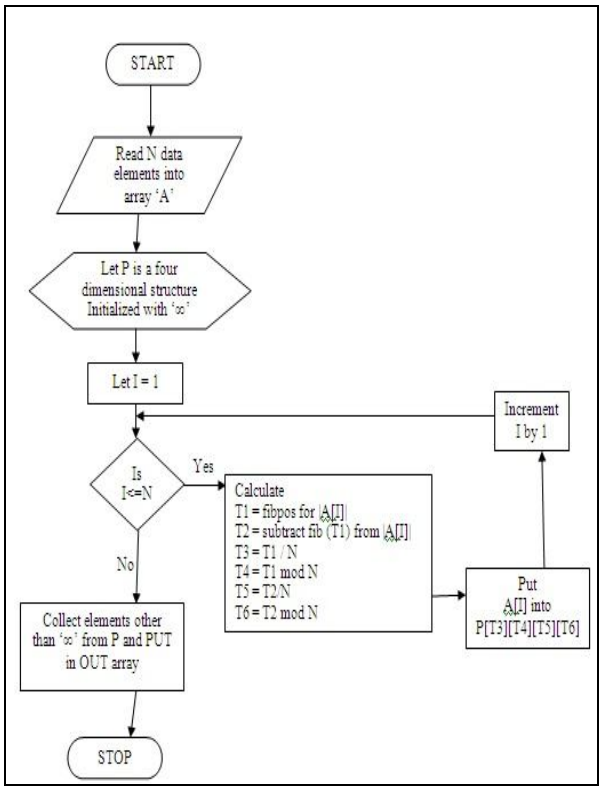

Figure 2. Logical Flow of FDM Sort

3.1 Sample Code explains and performs the sorting procedure

```
#include <conio.h>
#define n 8
#define size 25
long fib(int);
voidfibgen(long*);
void process(long,long*,long*);
long f[size];
void output(int,int,int,int,long);
long out[n][n][n][n];
longopdiv(long);
void main()
{
  //long
a[]={3760,3751,3761,3750,3759,3752,3743,375
4};
long a[]={-3780,-3740,-3766,-3750,-11,-3770,-
3720,-3710};
long s[n],t;
inti,j,k,l;
long p[n][6];
clrscr();
   //generate fib series
fibgen(&f[0]);
printf("\n----------------------------------------------
-\n");
for( j=0;j<size;j++)
    {
printf("%7d",j);
    }
printf("\n----------------------------------------------
-\n");
for( j=0;j<size;j++)
    {
printf("%7ld",f[j]);
    }
printf("\n----------------------------------------------
---");
  /*output array*/
for(i=0;i<n;i++)
   {
for(j=0;j<n;j++)
for(k=0;k<n;k++)
for(l=0;l<n;l++)
 out[i][j][k][l]=-9999;
   }
printf("\n----------------------------------------------
---");
printf("\nval\tp\t dif \t p/n \t p mod n \t dif/n \t dif
mod n");
printf("\n----------------------------------------------
---");

for(i=0;i<n;i++)
   {
process(abs(a[i]),&p[i][0],&p[i][1]);
p[i][2]=opdiv(p[i][0]);
p[i][3]=p[i][0]%n;
```


Figure 2. Logical Flow of FDM Sort

```
p[i][4]=opdiv(p[i][1]);
p[i][5]=p[i][1]%n;

printf("\n%ld\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld",a[i]
,p[i][0],p[i][1],p[i][2],p[i][3],p[i][4],p[i][5]);
    //distribute to output array
    output(p[i][2],p[i][3],p[i][4],p[i][5],a[i]);
  }
printf("\n-----------------------------------------------
---");
  //collect and store in sort array
for(t=0,i=0;i<n;i++)
  {
for(j=0;j<n;j++)
for(k=0;k<n;k++)
for(l=0;l<n;l++)
 {  if (out[i][j][k][l]!=-9999)
    {  s[t++]=out[i][j][k][l];
     }
  }
  }
printf("\n sorted array is \n");
for(i=0;i<n;i++)
  {
printf("\t%ld",s[i]);
  }
getch();
}
void output(inti,intj,intk,intl,longval)
{
if(out[i][j][k][l]==-9999)
out[i][j][k][l]=val;
}
voidfibgen(long *p)
{
long i,f1=-1,f2=1,c;
for(i=0;i<size;i++)
 {
   c=f1+f2;
p[i]= c;
   f1=f2;
   f2=c;
 }
}
void process(long val,long*k,long *d)
{
inti=0;
while(1)
 {   //locate in fib series
if ( f[i] >val)
      {
  *k=i-1;
  *d=val-f[i-1];
 break;
 }
 i++;
 }
}
long fib(int a)
{
if (a<1)
return 0;
else
if (a<=2)
return 1;
else
return (fib(a-1)+fib(a-2));
}
longopdiv(long val)
{
val=val/n;
returnval;
}
```

3.2 Sample processing to be performed by the sorting procedure using only positive set of numeric values

Table 1. First Data Sample in unsorted order (before sorting)

3760	**3751**	**3761**	**3750**	**3759**	**3752**	**3743**	**3754**

Table 2. Sorting Procedure Analysis for given First Sample Data

val	p	dif	p/n	p%n	dif/n	dif % n
3760	18	1176	2	2	147	0
3751	18	1167	2	2	145	7
3761	18	1177	2	2	147	1
3750	18	1166	2	2	145	6
3759	18	1175	2	2	146	7
3752	18	1168	2	2	146	0
3743	18	1159	2	2	144	7
3754	18	1170	2	2	146	2

Table 3. First Data Sample in ascending order (after sorting)

3743	**3750**	**3751**	**3752**	**3754**	**3759**	**3760**	**3761**

3.3 Sample processing to be performed by the sorting procedure using only negative set of numeric values

Table 4. Second Data Sample in unsorted order (before sorting)

-3780	**-3740**	**-3766**	**-3750**	**-11**	**-3770**	**-3720**	**-3710**

Table 5. Sorting Procedure Analysis for given Second Sample Data

val	p	dif	p/n	p%n	dif/n	dif % n
-3780	18	1196	2	2	149	4
-3740	18	1156	2	2	144	4
-3766	18	1182	2	2	147	6
-3750	18	1166	2	2	145	6
-11	6	3	0	6	0	3
-3770	18	1186	2	2	148	2
-3720	18	1136	2	2	142	0
-3710	18	1126	2	2	140	6

Table 6. Second Data Sample (after sorting)

-11	**-3710**	**-3720**	**-3740**	**-3750**	**-3766**	**-3770**	**-3780**

3.4Analysis of Algorithm:

The FDM sort distributes each element of source into external data structure by identifying the relative position in Fibonacci sequence and differential value with the Fibonacci value at that position. Later calculate the appropriate distribute point using division and modulus operations. Hence there is no key comparison. This only involves lookup, division and modulus operations. We assume that time required for a single operation is ŧ. Then total time required in FDM sort is then estimated as below

For initialization of external data structure P (Step-2 of Algorithm) T1=n*ŧ

For n elements distribution(Step-3 of Algorithm)

T2=n*7*ŧ

For n elements collection and put in output array (Step 4 of Algorithm)

T3=n*ŧ

Hence, the total time required to sort N number of elements is

T (N) = T1+T2+T3
= n*ŧ+7n*ŧ+n*ŧ
= (n+7n+n) ŧ
= 9nŧ

The observation of sorting procedure concludes as follows...

i) The algorithm is capable for either positive or negative values but not capable for set of both.
ii) The performance is not dependent on digits significance that mostly influence in case of radix and bucket sort.
iii) The algorithm is stable that is an element with the same value appears in the output array in the same order as they appear in the inputarray.
iv) This is not an in place sorting algorithm. We need external space to store the output.
v) This is a kind of distributed algorithm because elements are distributed and collected into output array. It has no comparison between elements.
vi) The operational performance is equivalent for each element to sort even in the presence of a small number of elements or more number of elements.

Comparison with other distribution sorting methods*:*

The comparison of various distribution algorithms are carry out with different input data sizes. Here the performance of radix, bucket, counting and FDM sorts are made known here with different input list of sizes.

Table 7: Comparison of various sorting algorithms [8]

Alogrithm	Time complexity
Radix sort	T(n)=(a+d+2e)*c*n
Bucket sort	T(n)=c*n+10k(2-1/n)
Counting sort	T(n)=5+(3c+8)*n
FDM sort	T(n)=9*n

The following chart depicts various sorting procedures pattern in processing various data sizes with proportionate to time.

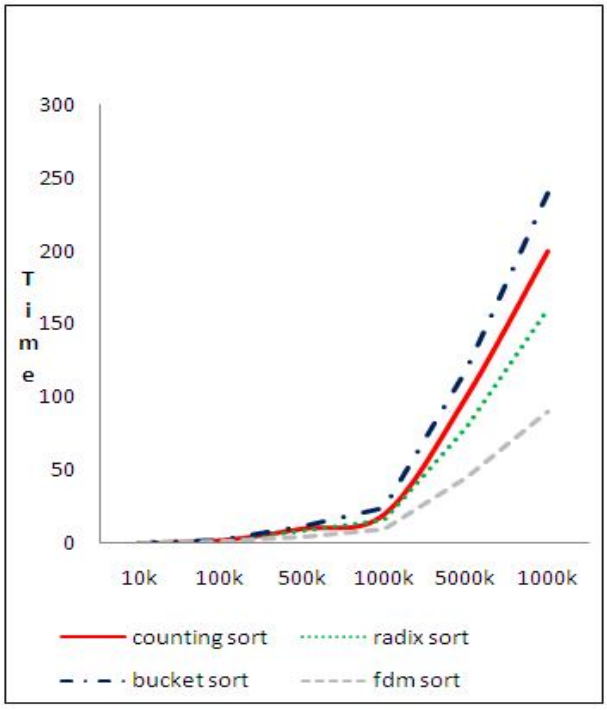

Figure 1. Performance of various Distribution sorting algorithms

4.Proof

4.1 Theorem:

Given a set T of unique integers in Z such that T= $\{t_1 \neq t_2 \ldots t_{n-1} \neq t_n\}$ and given a function F (e) such that:

$$F(e) \equiv \left\{ \begin{array}{l} X = \text{fibpos of } e \\ Y = e - fib(X) \\ P = X \text{ div } n \\ Q = X \bmod n \\ R = Y \text{ div } n \\ S = Y \bmod n \end{array} \right\}$$

Where n > 0 , then the resulting image set R of original quadruple is $R=\{(P_1,Q_1,R_1,S_1),(P_2,Q_2,R_2,S_2), \ldots ,(P_{n-1},Q_{n-1},R_{n-1},S_{n-1}),(P_n,Q_n,R_n,S_n)\}$ from the pre-image set T is an injective mapping F: T -> R

4.2 Proof:

Proof by Contradiction: Proceed with the assumption that given T as described before that ∀T, ∃(x,y)∈T(for all T there exists x,y is element of T)such that F:X = F:Y when x≠y, or that the function F:T→R is a non-injective mapping.
The definition of injective is "A function F is one-to-one if and only if f(x)=f(y) implies that x=y for all x and y in the domain of the function".

∀(x,y)∈F:X→Y,where (f(x)=f(y))ↄ(x=y) is injective.

By taking the contrapositive of this definition, it is restated as "A function f is one-to-one if and only if f(x) = f(y) whenever x≠y in the domain of the function." (Rosen 1991, p.57)

∀(x,y)∈F:X→Y,where ¬(x=y)ↄ¬(f(x)=f(y))is injective.

//to read as for all (x,y) belongs to f:x->y where not(x=y) subset of not(f(x)=f(y)) is injective

By taking the contrapositive of definition, it can be restated as "A function f is one-to-one if and only if

f(x) ≠f(y) whenever x≠y .(Rosen 1991,p.57)

Using the contrapositive of the definition of an injective function, it is readily clear that the mapping F:T→R is not injective if there are at least two integers k1 and k2 such that by the mapping function F,(p1,q1,r1,s1)=(p2,q2,r2,s2) in R. This is assumed to be true, as a non-injective mapping function.
By the definition of T, all the integers are unique in Z, so the integers have the property that for any two integers k, k' ∈ T so that k≠ k'.

Take the case of k:

X_k=*fibposof k;* Y_k=*k–fib*(X_k)*;*
P_k=X_k *div n;* *Qk*=X_k *mod* n; R_k=Y_k div n;
S_k=Y_k mod n

Take the case of k' = k+c where 0<c<n:

X_k=fibposof k'; Y_k=k'–fib(X_k);
P_k=X_k div n; Qk=X_k mod n; R_k=Y_k div n;
S_k=Y_k mod n
So that after substituting k' with k+c

X_k=fibposof (k+c); Y_k=(k+c)–fib(X_k);
P_k=X_k div n;Q_k=X_k mod n; R_k=Y_k div n;
S_k=Y_k mod n

Whenever the integers k, k' are unique then with the reference to Fibonacci sequence $(X_k, Y_k)\neq(X_{k'}, Y_{k'})$

Now consider the remaining operations

P_k=X_k div n; Q_k=X_k mod n;
R_k=Y_k div n; S_k=Y_k mod n;

Take the case of r:
P_r=r div n; Q_r=r mod n;
P_r =d_r where $d_r \geq 0$; Q_r=$M_r$where $0 \leq M_r \geq n$;

Take r'= r+c where 0<c<n:
P_r=r' div n; Q_r=r' mod n;
P_r=(r+c) div n; Q_r=(r+c) mod n;
P_r=(r div n)+(c div n);

Q_r=(r mod n)+(c mod n);
P_r= d_r +r where d_r, r≥0;
Q_r= M_{r+}c where 0≤M_r≥n;
So then that for r = r' that F(r)≠ F(r').

But this is for r'= r + c where 0<c<n. Take the case when c≥n, using the same definition for r, take r'=r+c where c=n.

P_r=r' div n; Q_r=r' mod n;
P_r=(r+n) div n; Q_r=(r+n) mod n;
P_r=(r div n)+(n div n); Q_r=(r mod n)+(n mod n);
P_r= d_r + 1 where d_r≥0; Q_r= M_{r+}0 where 0≤M_r≥n;
P_r= d_r +1 where d_r≥0; Q_r= $M_r$where 0≤M_r≥n;

So then that for r = r' that F(r) ≠ F (r').
For each case of the form r'=r+c, where c<n and c≥n, if r=r' then F(r)=F(r') which contradicts with the original assumption that ∃(x,y)∈T so that F:X = F:Y when x≠y.

So the occurrence of $d_{r=}$ $d'_r$and$M_{r=}$ M'_rsuch that r=r' is never true and will never occur, which is the only possibility for the mapping F:T→R to be non-injective.

Thus it follows that no two quadruple in R formed from any two integers in T by F would ever be equal. Since this is the only counter example for the definition of F(x) to be an injective mapping, then the only conclusion is that F:T→R must be an injective mapping under F(x).

5.CONCLUSION:

The initial works on this is made on numerical elements and reduce performance bottlenecks that are ubiquitous in distribution approaches. Further works need to concentrate on source that must be diverge, sparse and utilize utmost possible limited external space.

REFERENCES

[1] H.S.M.Coxter "The Golden Section, Phyllotaxis and Wythoff's Game" Scripta atheatica 1:1 (1953) pp. 135-143.

[2] J.Devita,"Fibonacci,Insects and Flowers","Fibonacci Quarterly 16 (August, 1978) 315-317.

[3] John Atkins, Robert Geist , " Fibonacci Numbers and Computer Algorithms".

[4] Janvier Nzeutchap ," Young –Fibonacci insertion, tableauhedron and Kostka numbers",Journal of Combinatorial Theory, Series A 116(2009) 143-167.

[5] P.S. Stevens ,Patterns in Nature, Atlantic Monthly Press, Little Brown and Company, Bostan, 1974.

[6] William F. Gilreath, "Hash sort: A Linear Time Complexity Multiple-Dimensional sort Algorithm" originally entitled "making a hash of sorts"", Aug-2004.

[7] Donald. E.Knuth," The Art of Computer Programming Vol:3 Sorting and Searching",Addison-Wesley ,1973.

[8] Debasis Samanta,"Classic Data Structures 2nd edition", PHI Learning private Limited,2012.

[9] P. Chitti Babu and S.Ramakrishna, "Assessment of Maintainability factor", International Journal of Computer Science Engineering and Information Technology Research Vol. 3, Issue.3,29-42, 2013

AUTHORS

Mr. C.Siva Krishnaiah is currenty working as Assistant Professor, Annamacharya PG College of Computer Studies, Rajampet, AndhraPradesh. He received hisMaster's degree in in Computer Applications (MCA) from Madurai Kamaraj University,Madurai. He has completed M.Phil in Computer Science fromMadurai KamarajUniversity,Madurai and M.Tech in Computer Science & Engineering from Acharya NagarjunaUniveristy, Guntur, India. He has a total of12 years of experience.

Dr. P. ChittiBabu is currently working as Professor & Principal, Annamacharya PG College of Computer Studies, Rajampet,AndhraPradesh. He received his Master's Degree in Computer Applications (MCA) from SV University, Tirupathi. He has completed M.Phil in Computer Science from Alagappa University and M.Tech in Computer Science & Engineering fromAcharya NagarjunaUniveristy, Guntur, India. He has successfully received his Ph.D (Computer Science) from S.V. University. He has a total of 15 yearsofexperience. He has published 25 papers in International and National Journals. He is a life member of CSI, IACSIT, IAENG and ISTE. Presently he is guiding 2 students for research under Ph.DProgramme