EXTERNAL DATA PREPROCESSING FOR EFFICIENT SORTING

S.Hrushikesava Raju[1], T. Swarna Latha[2]

[1]Associate Professor, Dept. of CSE, SIETK, Puttur, ***hkesavaraju@gmail.com***
[2]Professor, Dept. of CSE, St.Anns Engineering college, Chirala, ***proftswarnalatha@gmail.com***

Abstract

*Paper presents external sorting using data preprocessing. Generally, huge data of any organization possess data redundancy, noise and data inconsistency. To eliminate, Data preprocessing should be performed on raw data, then sorting technique is applied on it. Data preprocessing includes many methods such as data cleaning, data integration, data transformation and data reduction. Depending on the complexity of given data, these methods are taken and applied on raw data in order to produce quality of data. Then, external sorting is applied. The external sorting now takes the number of passes less than actual passes log $_B$ (N/M) + 1 for B – way external merge sorting, and number of Input / Outputs less than 2*N* (log $_B$ (N/M) + 1) of Input / Outputs and also involve least number of runs compared to actual basic external sorting.*

Keywords: *data preprocessing, external sorting, Data cleaning, passes, Inputs / Outputs, and runs.*

---***---

1. INTRODUCTION

In real world, most data collected should be huge and that consists of lot of irregularities in terms of missing data, noise, or even outliers. This data doesn't possess quality of information. A data mining technique called Data Preprocessing is required in order to get quality data by removing such irregularities. The data Preprocessing technique has four methods and these are used appropriately to eliminate particular complexities that each method can remove. Those methods are Data cleaning, Data Integration and Transformation, Data reduction, and Data Discretization and Summarization.

The first method, Data cleaning is used when incomplete, noise, or any outliers exist in the data that can be removed using any of binning, clustering, and regression techniques. Second is Data integration and transformation is used when the data set contains objects with many values or different objects with same value and the data is not in required interval or range and this can be eliminated by processing using care in case of integration and use any of smoothing, attribute/feature construction, normalization, or aggregation in case of transformation. The third, data reduction is when the data set is high dimensional or in large volume, and this can be avoided by using any of dimensionality reduction, numerosity reduction or data compression in order to output the reduced size of that data set which produce the same results. The data discretization and summarization is used when data is in continuous and that can be broken into intervals using either top down or bottom constructions.

The following table shows when each stage is required and what that method will do.

Table 1.1: Data preprocessing method's regularities and their output

Method name	**Irregularity**	**output**
Data cleaning	Incomplete, noise, inconsistent, missing	Quality data Before integration
Data Integration and transformation	Object identity problem	Quality data with care taken
Data reduction	Data set is high dimensional	Reduced size
Data Dicretization and summerization	Data continuous	Simplified data sets

The methods of the data preprocessing are conveniently applied depending on the complexity of the original data set. This technique when applied to external sorting yields time complexities less than their actual time complexities. Although several approaches such as double buffering, clustered or un clustered B$^+$-tree are used, that take lot of memory while coding those structures. The list of external techniques is listed in table 1.2 along with their complexities and purposes.

Table 1.2: External sorting Techniques

Technique	Purpose	Overhead or complexity
Double buffering	Minimize I/O operations	Additional buffer is maintained for each input and output buffer
Key Sorting	Keys are small compared to records	Each record associated the key cause expensive.
Replacement Selection	Makes heap	Involves many swappings from root with last node and then discards last node value, reconstructs heap until one element remains. Its Time complexity – O(n * log n)
Clustered B+ - tree	Index llows to search for record	Sorts by traversing the leaf pages
un clustered B+ - tree	Index allows to search for record	Sorts by data records. Additional cost incurred for each page retrieved just once.

All these external sorting techniques don't minimize the disk Input / Output time efficiently. Each technique has their own drawbacks. Thus, These drawbacks also because of data redundancy and replication on each page or tape. This leads to redundancy after sorting runs in each phase. To avoid much time complexities or Minimize I/ O costs, data preprocessing is necessary before sorting on external storage devices such as tapes, pages or disks etc. The advantage of performing data preprocessing is redundant data is eliminated from tapes or pages, sorting data doesn't contain redundancy which also minimize the I/O costs in sorting.

2. RELATED WORK

According to [14],[15] and [1,2], the various external sorting techniques although performing external sorting that achieved with a variety of overheads. According to [9], certain types of lemmas are used to achieve efficient external sorting but it works on only one disk model although it takes less Input / Output operations than normal merge sort. To overcome the overheads of various external sorting techniques and also external sorting using 3 lemmas, data preprocessing [4,5] is proposed before external sorting is used. The external sortings are categorized into 2 types importantly. They are k-way external sorting and poly phase merge sorting for k-way merge sorting. According to Mark Allen Weiss, the external sorting applied on the data although that r of disk accesses or inputs / outputs costs huge compared to disk accesses on data without redundancy.

The time complexities of k-way merging and poly phase merging on the data that involves redundancy is in the table 2.1. The variables here are k denote k-way sorting, N denote number of items on initial tape, and M is initial run size.

The time complexities or overheads incurred for the same external sorting strategies that don't involve redundancy are found less than actual complexities for the data that involve redundancy by some examples in Experimental results.

Table 2.1: Time Complexities for external sorting strategies

Method	Tapes required	# of passes
K-way external sorting	2 * k	Log $_k$ N/M +1
Poly phase merging	k + 1	Depends on how large the data size on initial tape

3. PROPOSED WORK

In this, Data preprocessing is an important task that removes redundancy by using a variable length record. This record avoids loss of data and is used to represent the data only once although it maintains the number of occurrences of that item.

This data preprocessing also used to reduce the number of runs involved in the merge during each pass when the data possess redundancy hugely. A method of data preprocessing called Data Cleaning is used to eliminate data redundancy, noise, and inconsistency that exist in data available on initial tape or page. The advantages of data preprocessing is shown in table 3.1 as follows according to type of data.

Table 3.1: Data preprocessing uses according to type of data

Type of data	usefulness
Numeric data with redundancy	Eliminate redundancy without loss of data by using variable length record.
Alphanumeric data	It sorts numeric data separately, and alphabets separately on each of alternative tapes.
Strings	Eliminates redundancy by using string type of variable length record.

In all above types, data preprocessing reduce the number of disk accesses or Input / output cost in terms of eliminating redundancy. Data Preprocessing supposed be applied on the data before sorting results many benefits greatly that leads to perform external sorting efficiently. Here, data preprocessing algorithm or pseudo code is given for each type of data and also algorithm is defined for external sorting.

A. Data Preprocessing: The separate methods are defined based on type of data such as integer, alphanumeric, and strings.

a. Pseudo code for Numeric data with redundancy:

```
Void NdataPreprocessor(int data[][], int size)

    1.  define variable length record that contain numeric key
        along with count.

        typedef struct Nrecord{
        int key, count;
        }

        Nrecord *arr; // it stores various duplicated numbers
        along with count from arr[0] to arr[i] where i denote i
        number of duplicated elements exists.

        declare int array – int *k, also int *hh used to store
        unique key values and index k is used;

        declare index h for k array with initial value 0, p
        maintains count for each duplicated element

        and is 0 initially. define z= r * c;

    2.  compare first key with rest of the keys in order to test
        for redundancy. This is repeated for rest of keys with
        later elements.

        for i=0 to number of rows i.e r and i++ for every next
        step

        for j=0 to number of columns i.e c and j++ for every
        next step

        {  k[h]=data[i][j]; h++; }

        for(i=0;i< r * c; i++) // testing each key with later
        elements

        {

        key=k[i];  // key is temporary int variable

        for(j=1;j<r*c;j++) {  // traverse the total array

        if(key == k[j]) { ++p;  // counts redundant item

if(p>1) // item removed from second copy onwards  {

free(int(k[j])  // removes duplicated memory

z--; // int variable z maintains up to date array size  }

else continue; }

if(p!=0 && p!=1)

{
arr[i]=(key, ++p); // redundant item value, count
hh[k1]=arr[i].key;  k1++; }
else { hh[k1]=k[i]; k1++;
display element is not redundant }

    3.  Now, data resulted in hh int array doesn't contain
        duplication for any element. This also represents data
        preprocessing for Numerical data.
```

b. Data Preprocessing for Alphanumeric data:

```
Void ANDataPreprocessor(char data[][],int size)
1.        Declare two arrays inta for integer data and chara for
character data. int *inta; char *chara;
2.
Declare integer k array – int *k to store two dimensional data.
declare indexes h, i, j

a)        separate the given data such that inta stores only
numeric data, and chara array stores only character data. It
involves 2 steps.

b)        Storing given data in single character array


for(i=0;i<r; i++)
for(j=0;j<c;j++)
 k[h]=data[i][j];
c)        separate data in single dimensional array into
appropriate arrays inta, and chara.

indexes d for chara and j for inta are used are 0 intially.
for(i=0;i<r*c;i++)
if(k[i]>=a&&k[i]<=z||k[i]>=A && k[i]<=Z)
{         chara[d]=k[i];  d++; }
          else if(k[i]>= 36 && k[i]<=45) {
           inta[j]=k[i];  j++; }
C)        Eliminate data redundancy in inta array by calling
NDataPreprocessor() and also in chara data by calling the
following code.
z=r * c; // data set size
for(i=0;i< r * c; i++) //
{
key=k[i]; // key is char variable
for(j=1;j<r*c;j++) {  // traverse the total array
if(key == k[j]) { ++p;  // counts redundant item
if(p>1) {
free(int(k[j])  // removes duplicated memory
z--;   // maintains up to date array size      }
else continue; }
if(p!=0 && p!=1) { arr[i]=(key, ++p); // redundant item value,
count where arr record contains char type key.
```

```
gg[k1]=arr[i]. key; k1++; where gg int array stores unque char keys and used its index k1 and is 0 initially. }

else { gg[k1]=k[i]; k1++;
      display element is not redundant  }
```

3. Now, two data arrays int array hh and char array gg resulted doesn't contain duplication for any element. This also represents output of data preprocessing for both Numerical and character data.

c. Data Preprocessing for String Data:

It eliminates duplicated copy of every string. It also uses variable length record that contains string as key and also number of occurrences of it.

```
typedef struct Srecord
{
String *s;   int count; };
```

The Pseudo code is defined as follows:

```
Void SDataPreprocessor(String data[][], int size)
```

1. Declare string array- String *ss[r * c]; // variable length strings are accessed by ss. r * c is data set size. Also String *as[z] where z value is resultant set size and is r * c initially.Declare indexes i, j in order to compare first key with rest, and second with third onwards, and third with fourth onwards, and so on.

2. Eliminate duplication in string data.

a. store strings in two dimensional form into a single dimensional form.

b.

```
for(i=0;i<r;i++)
for(j=0;j<c;j++)
{
ss[k]=data[i][j];  // initial k is 0
```

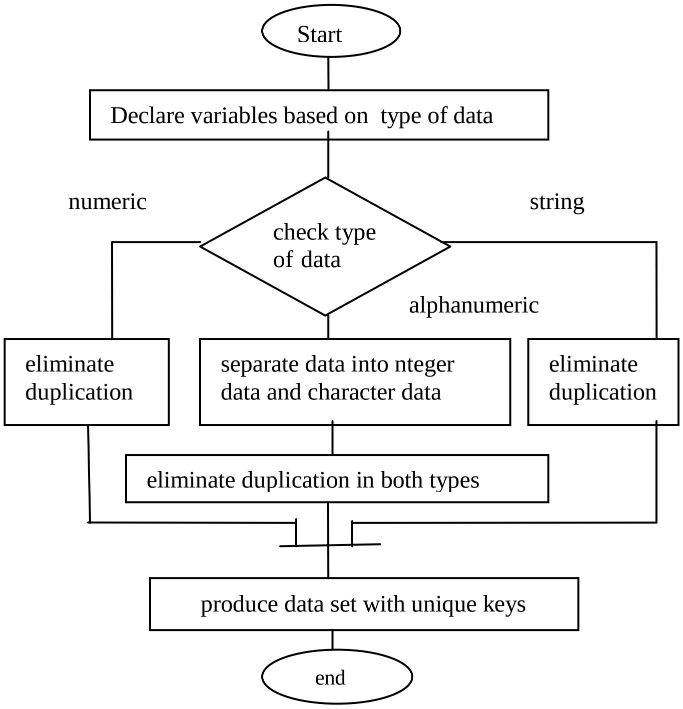
c. duplication of strings that are maintained only once in the data set.

```
for(i=0;i< r * c; i++) //
{
key=ss[i]; where key is also variable length string.
for(j=1;j<r*c;j++) {  // traverse the total array
if(key == ss[j]) { ++p;  // counts redundant item
if(p>1)   // remove from second  copy onwards {
free(SizeOf(ss[j])  // removes duplicated memory
z--; // maintains up to date array size    }
else continue; }
if(p==0 || p==1 and m<z) //m is index and is 0
{ as[m]=ss[i];
  m++ ; //unique keys are stored in as string array}
if(p!=0 && p!=1) {
arr[i]=(key, ++p); // redundant item value, count
else element is not redundant
as[m]=arr[i].key; m++; // index incremented to store next key
}
```

Flow Chart for Data Preprocessing:

It shows the flow of actions in data preprocessing based on type of data. This module in flow chart takes the data first, then applies appropriate data preprocessing method depending on type of data provided to sort.

Moreover, It is a Graphical technique that clearly conveys information and its meaning to the end user who even doesn't know about programming. The following denotes flow chart for data Preprocessing module which can be applied prior to sorting.


Start
Declare variables based on type of data
numeric
string
check type of data
alphanumeric
eliminate duplication
separate data into nteger data and character data
eliminate duplication
eliminate duplication in both types
produce data set with unique keys
end

Flow Graph3.1: Data Preprocessing Steps

B. Algorithm for external sorting:

It provides the pseudo code that accomplishes the external sorting for m-way merge sorting. This sort works same line K-way merge sort but it involves least number of runs for the data that initially have lot of redundancy.

Algorithm **B-way Externalsortnig**(dataset)

Input: put the items of dataset on first tape or page of inputs. 2 * B tapes or pages are used.

output: sorted data placed on single tape or page

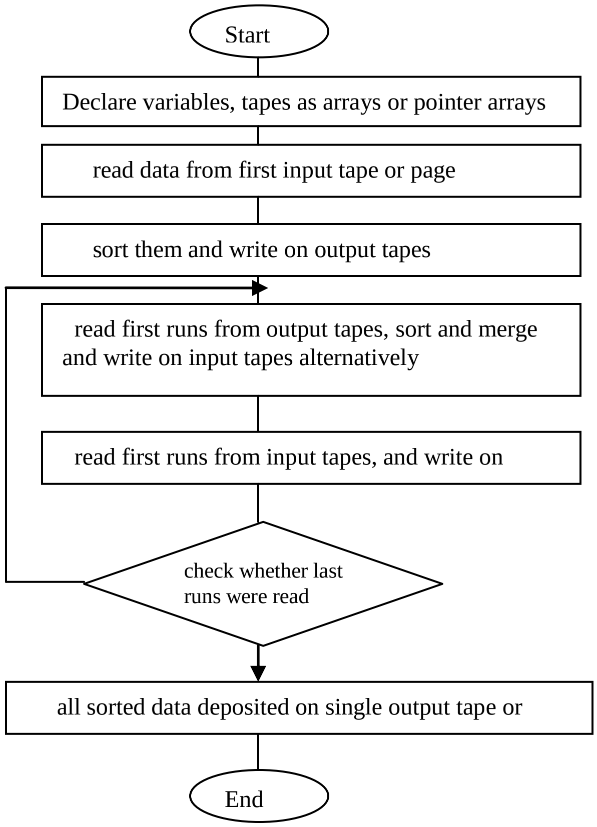
1. Take data on first tape or page of size M which is main memory size successively till data size is reached.
2. Sort them internally using either Quick sort or Merge sort and Write them on output tapes alternatively.
3. Read first runs from all output tapes or pages, merge and sort them internally, and write it on input tape.

Similarly, Read second runs on all output tapes and write them on input tape alternatively on input tapes till last runs are read from all output tapes.

4. Read first runs from input tapes, merge and sort them using sorting algorithm and write on output tapes alternatively from second run to last runs that are read from input tapes.
5. repeat 2 to 4 steps till total data comes on single output tape.

Advantages: Although it yields same number of passes in some cases, the number of disk accesses or input and output costs are reduced greatly in terms of runs.

Flow chart for efficient sorting:


Start
Declare variables, tapes as arrays or pointer arrays
read data from first input tape or page
sort them and write on output tapes
read first runs from output tapes, sort and merge and write on input tapes alternatively
read first runs from input tapes, and write on
check whether last runs were read
all sorted data deposited on single output tape or
End

Flow Graph 3.2 : External Sorting Steps

4. Experimental results with Examples

This presents a table which shows the number of runs incurred before data preprocessing and after data preprocessing. From this, how the numbers of disk accesses are involved or the complexity involved is affected in terms of input and output costs and runs.

Example1: Consider the data set 10 3 3 7 1 1 1 78 2 2 2 2 3 3 3.

The purpose of taking higher way of merge sorting is to reduce number of passes. Examples include 2-way merging takes 4 passes, 3-way merging takes 3-passes, and so on. Consider T_i and T_o are input and output tapes from which both read and write are possible.

a. With Redundancy:

Step1: Initial Run construction pass – assume main memory size is 3

T_i1 10 3 3 7 1 1 1 78 2 2 2 2 3 3 3
T_i2
T_i3
T_o1
T_o2
T_o3

Step2: read data of run size 3, sort and write on output tapes alternatively.

T_i1
T_i2
T_i3
T_o1 3 3 10 2 2 2
T_o2 1 1 7 3 3 3
T_o3 1 2 78

Step3: Read first runs, merge and sort them, and write it on input tape. Next, read second runs and merge and sort them, write on input tapes alternatively till lazst runs from output tapes are read.

T_i1 1 1 1 2 3 3 7 10 78
T_i2 2 2 2 3 3 3
T_i3
T_o1
T_o2
T_o3

Step4: Read first runs from input tapes, merge and sort them, and write on output tape. Read second runs from input tapes, sort and merge them and write on output tapes alternatively till last runs were read from output tapes.

T_i1
T_i2
T_i3
T_o1 1 1 1 2 2 2 2 3 3 3 3 3 7 10 78
T_o2
T_o3

Factor	Sorting on Data with redundancy	Sorting on Data after data preprocessing
Input and output costs or number of disk accesses	(15 + 15+15 + 15) * 2 = 120 accesses = 2*N* (log$_B$ (N/M) + 1) =2 * 15 * 4	2 * 6 * 3 = 36 accesses
Number of runs	5+2+1 excluding initial pass	2 + 1 excluding initial pass
Number of passes	3 + initial run pass = 4	2 + initial pass =3

There are four passes incurred for the above data set that involve redundancy.

b. Without Redundancy:

First, data preprocessing applied on original data which is huge and involves inconsistency, and noise generally. So, data preprocessing is used to eliminate such things from original data and produces data set of items: 10 3 7 1 78 2

Step1: Initial run construction pass – run size is 3.

T_i1 10 3 7 1 78 2
T_i2
T_i3
T_o1
T_o2
T_o3

Step2: read data input tape according to run size 3, sort and write on output tapes alternatively.

T_i1
T_i2
T_i3
T_o1 3 7 10
T_o2 1 2 78
T_o3

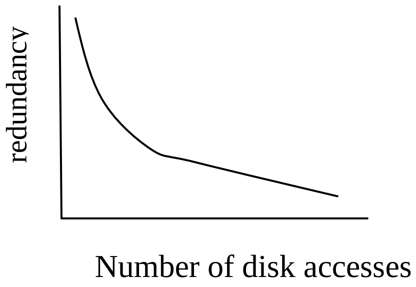
Step3: read data from output tapes, merge and sort them, and write on input tape alternatively.

T_i1 1 2 3 7 10 78
T_i2
T_i3
T_o1
T_o2
T_o3

Only three passes are required to sort data on external device for the data without redundancy.

The following table shows the significance difference between data with redundancy and data preprocessing whenever to sort data using B-way external sorting.

The following graph denotes that if the large data set contains lot of redundancy, then the number of disk accesses are reduced or minimized by applying data preprocessing. The data set that possess lot of redundancy is inversely proportional to number of accesses or input and output costs, runs and even number of passes.



Graph4.1: Redundancy versus Number of accesses relationship

CONCLUSIONS

This concludes that as data redundancy exists as increasing. Data preprocessing module helps to reduce the number of disk accesses or number of input and output costs, number of runs, and even number of passes for B-way external merge sort. The data preprocessing is flexible to work on the data of any data type. This data preprocessing module also helps to avoid loss of data by defining a record which contain item value and count of it for each duplicated element in the data set. This work can be enhanced in future such that it can also be implemented on individual items when they are of records or some complex data types and can be allowed to sort them efficiently.

REFERENCES:

[1] Chapter7, Data Structures and Algorithm Analysis in C++ by Mark Allen Weiss.
[2] Chapter7, Data Structures and Algorithm Analysis in Java by Mark Allen Weiss.
[3] Sorting, Data Structures and Algorithms Alfred V. Aho, John E. HopCroft and Jelfrey D. Ullman, Addison –Wesley, 1983.
[4] Data Preprocessing, Data Mining Principles and Techniques by Micheline Kamber and Jiawei Han.
[5] Margaret H Dunham, Data Mining Introductory and Advanced Topics, Pearson Education, 2e, 2006.
[6] Sam Anahory and Dennis Murry, “Data Warehousing in the Real World”, Pearson Education, 2003.

[7] D. E. Knuth (1985), *Sorting and Searching, The Art of Computer Programming, Vol. 3*, Addison –Wesley, Reading, MA, (1985).

[8]] AV88.pdf, Input and Output Complexity of Sorting and related problems, Algorithms and Data Structures by Alok Aggarwal and Jeffrey Scott Vitter.

[9] An efficient External Sorting Algorithm, pp.159 – 163, by Leu, , Fang-Cheng; Tsai, Yin-Te; Tang, Chuan Yi ,Information Processing Letters 75 2000.

[10] Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems), Ian H. Witten, Eibe Frank, Morgan Kaufmann, 2005.

[11] Pt03.pdf, Introduction to Data Mining, part3: Data Preprocessing, Zhi – Hua Zhou, Dept. of CSE, Nanjing University, Spring 2012.

[12]www.cs.uiuc.edu/homes/hanj/cs412/bk3.../ **03Preprocessing.ppt.**

[13]*ww.cs.gsu.edu/~cscyqz/courses/dm/slides/**ch02.ppt.***

[14] *inst.eecs.berkeley.edu/~cs186/fa06/lecs/ **05Sorting.ppt.***

[15] *www.cs.rutgers.edu/~muthu/**lec9-04.ppt.***

[16] data.ppt.pdf, Introduction to Data Mining: Data Preprocessing by Chiara Rebso, KDD- LAB, ISTI – CNR, Pisa, Italy.

BIOGRAPHIES:

S. HrushiKesava Raju, working as Associate Professor in the Dept. of CSE, SIETK, Narayanavanam Road, Puttur. He is persuing Ph.D from Raylaseema University in area Data Mining tasks on advanced Data Structures. His areas of interest are Data Mining, Data Structures, and Networks.

T. Swarna Latha, working as Professor in the Dept. of IT, St. Anns Engineering and Techology, Chirala. She had completed Ph.D from S.V. University in area Network Security. She is presently guiding many scholars in various disciplines.