Block Sorting of a Large File in External Storage by a 2-Component Key

K. P. Tan and L. S. Hsu
Department of Computer Science, National University of Singapore, Kent Ridge, Singapore 0511

This paper introduces a sorting method (a block sort embedded with another sorting algorithm) to sort a large volume of records in external storage by means of a 2-component numeric key consisting of key 1 and key 2. Based on the value of key 1, the data file is split into many subfiles. Each subfile contains records of common key 1, but with different key 2s. After sorting, all records are linked together by pointers to form a sort file. An index array keeps the list heads to these separate subfiles. The list structure in the file organization facilities updating of the file. This sorting method can be applied to a key of any number of digits. A subroutine of the methodology is illustrated in the Appendix.

INTRODUCTION

In general, when the main storage is inadequate to handle the sorting of a large volume of data, the data file is divided into several subfiles. Each subfile is sorted internally and stored on a magnetic tape. The tape files are then merged into a sort file by means of polyphase merge[1] or merging two linearly ordered files.[2] However, both merging methods involve a large number of read–write passes between magnetic tapes and the main memory storage, and are quite time-consuming. In the case of block sorting,[3] a file is split into ranked subfiles by dividing the key value of a sort key into certain ranges. Each subfile with keys falling into the same range is sorted separately. A complete sort file is obtained by joining all the sorted subfiles together. However, the capacity of the working storage in the main storage is the constraint to this approach.

Here, consider a small computer of limited main storage where a double precision feature is not available. A sorting method is introduced to sort a file of a large volume of lengthy records, in which the positive numeric key value of the sort key may be so long that it occupies two 16-bit words, say, a 6-digit key. This two-word key can be regarded as a two-component key, comprising two integers, denoted as key 1 and key 2. When key 1 consists of one, two, three or four digits from the left, key 2 will contain five, four, three or two digits counting from the right, respectively. Hereafter, this two-component key is also called compound key. Those key 2s which associate with the same key 1 value form a subfile. As the main storage is too small to keep the entire file, all records are read and stored as a disk file by entry sequence. To avoid frequent swapping of all data fields of one record with another in the sorting process, each record is attached with a pointer to denote the record number. Only the keys and the pointers are stripped off from the records of each subfile and are sorted separately in the main storage each time.

With the aid of the pointers all records can be linked in a list structure in a particular sequence of the sort key. Their physical locations remain unchanged in the disk storage. Therefore, whenever there is a deletion or an insertion of a record to the file, only the two pointers which are involved in the linkage of the two relevant records have to be amended. Saving storage space is another advantage of this sorting method. It requires just the disk space for the file, the working storage in the main memory for the biggest subfile and an array with a size equal to the order of key 1.

METHOD DESCRIPTION

In this method, records are stored in a disk file in the order in which they are read. The sort key is a compound key consisting of two parts, key 1 and key 2. An array, called key 1 address array, is maintained in the main memory. Its size should be compatible to the largest value of key 1. When a record is read and written to the disk, the ith array element stores the record number if the key value of key 1 is equal to i. Meanwhile the pointer in this

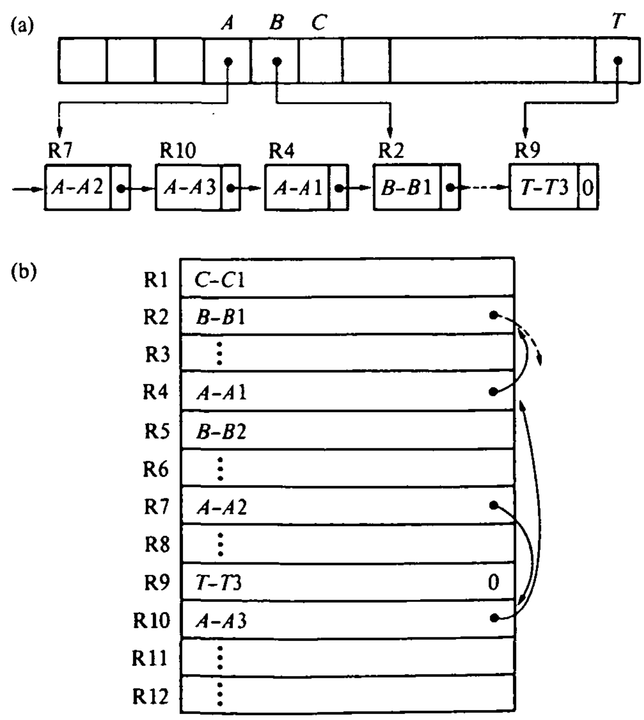

Figure 1. List organization of records with compound keys; (a) key 1 address array and the linkage of logical records; (b) linked records in storage.

CCC–0010–4620/82/0025–0327 $02.00

record stores the previous record number, which was kept in the ith array element, and whose compound key has the same key 1. As all records are stored in the disk, subfiles are also created. Each subfile contains records with the same key 1. They are linked together by their pointers. At this moment, each array element keeps the last record number of its corresponding subfile (i.e. the list head of the linked list). For instance, record R4 has a compound key A–$A1$ where key 1 has the value A and key 2 the value $A1$. When record R4 is read, the Ath element of the key 1 address array keeps the record number R4. This content is replaced by R7 when record R7, which is the next record containing the same key 1 value, is read in. On the other hand, the pointer of record R7 stores the previous record number, i.e. R4. As this process goes on, records R4, R7 and R10 contain the common key 1, say, A, and form a subfile as shown in Fig. 1(b). Thus the Ath array element keeps the list head, i.e. R10, and the pointer field of record R4 the list tail.

In the sorting process, only keys 2 and the pointers of the records of the Ath subfile are stripped off and sorted in the main storage. After sorting, the records of the subfile are relinked in the key 2 sequence (e.g. in ascending order). The new list head, say, record R7, which has the smallest key 2 value, would be stored in the corresponding Ath element of the key 1 address array (see Fig. 1(a) where, for simplicity, a logical record contains only the compound key and the pointer). The next subfile containing another common key 1, say B, where $B>A$, is sorted and then linked to the previous subfile A, and so on. The last record of the resulting sort file has a pointer with value 0 to indicate the end of the linked list. In this list structure, all records in the disk file remain in the same physical locations (see Fig. 1(b)). In this approach, only three passes of the entire file (i.e. write, read and rewrite) have to be processed. As the size of a subfile is practically small (see the section Hed Testing Results and Discussion), the bubble sort[1] can be adopted to sort each subfile. Consequently, a new sorting algorithm is developed. It is a novel combination[4] of the block sort and the bubble sort. In order to illustrate the algorithm of this sorting method, a subroutine with the imposed bubble sort is attached in the Appendix. For the purpose of demonstration, each record contains only three fields, i.e. key 1, key 2 and the pointer.

AN EXAMPLE: SORTING OF A 6-DIGIT KEY

As a simple example, consider a file of 20 records which would be sorted in a 16-bit word computer. The sort key contains six digits and is regarded as a compound key. If this compound key can be divided into two parts, key 1 and key 2, key 1 refers to the leftmost digit and key 2 to the other 5 digits on the right. Assume that the key 2 value does not exceed the limit of an one-word integer. Table 1 shows the key 1 address array of size equal to 9 and a disk file of 20 records, ignoring all other data fields except the compound key and the pointer in each record.

After sorting, the first element of the address array has a value (-4) indicating that record 4 has the smallest compound key value (i.e. 104016). The minus sign means there is only one record in the list whose key 1 has a value 1. This compound key is also illustrated in record 4 in the storage (i.e. 104016 -5). The pointer with value (-5)

Table 1. An example of the proposed block sorting; (a) key 1 address array; (b) 20 linked records in the disk file

(a) Index	Pointer	(b) Record number	Storage: Compound key	Storage: Pointer
1	−4	1	6 01641	7
2	−5	2	5 04812	10
3	16	3	8 01061	17
4	19	4	1 04016	−5
5	2	5	2 06353	−16
6	1	6	5 10164	12
7	−18	7	6 11997	−18
8	3	8	9 05080	9
9	14	9	9 12617	0
		10	5 05404	6
		11	4 08981	−2
		12	5 13440	−1
		13	8 08001	−14
		14	9 00772	8
		15	4 06349	11
		16	3 06568	20
		17	8 06841	13
		18	7 06508	−3
		19	4 02053	15
		20	3 11088	−19

denotes that the next smallest compound key should be in record 5 (i.e. 206353). On the other hand, the value ($+2$) in the array element 5 means that the record 2 has key 1 equal to 5 and the plus sign shows that there is more than one record containing the common key 1 (e.g. by tracing down records 2, 10, 6 and 12 in the disk storage). In record 9, the pointer with value (0) shows the end of the list. Starting with record 4 in the disk file (see Table 1(b)), all the 20 records are sorted by the 6-digit key in ascending order.

TESTING RESULTS AND DISCUSSION

Testing of this proposed sorting method was done on the IBM System 3 Model 10 computer. The compound key of a record is generated by a random number generator. It may consist of seven digits. Key 1 is chosen to represent the value of the first three digits on the left and key 2 represents the other four digits on the right. The size of the key 1 address array is chosen as 999. Eight sets of data files comprising 100, 500, 1000, 2000, 4000, 6000, 8000 and 10 000 records are sorted separately. The sorting time varies from 0.26 min to 38.75 min, respectively when the bubble sort is used to sort the subfiles (see Table 2). The Singleton sort[5], the fastest sort so far known, is

Table 2. Comparison in sorting time on the proposed block sort

Number of records	Proposed block sort: With bubble sort (min)	Proposed block sort: With Singleton sort (min)
100	0.26	0.27
500	1.18	1.27
1 000	3.03	3.15
2 000	6.20	6.67
4 000	13.62	13.90
6 000	21.38	21.23
8 000	29.47	28.72
10 000	38.75	36.88

then adopted to sort the subfiles on the same sets of data files. Since a subfile in this proposed block sort is always small, the Singleton sort does not improve the result much. For the case of 10 000 records it takes 36.88 min. only 5% faster than the block sort embedded with the bubble sort. In the case of a small volume of records, say, within 4000 records, the block sort, imposed with the bubble sort, takes even less time as the bubble sort's algorithm is much simpler. For a fixed number of records, say, 1000 records, the sorting time (with a bubble sort applied to subfiles) decreases obviously with increasing key 1 size (see Table 3). This is due to the decreasing

Table 3. Sorting time of 1000 records vs key 1 size (with bubble sort applied to the subfiles)

Key 1 size	Sorting time (min)
9	4.64
99	3.33
999	3.03
9999	2.52

number of records in a subfile. In other words, it is advantageous to choose a larger size for key 1 when key 1 is separated from a long compound key.

Because the sort file is merely a linked list, insertion of a new record is very simple. Based on the key 1 value of the new record, the list head to the required subfile from the key 1 address array is obtained and the linkage of the appropriate records changed. For deletion, the record deleted has to be disjoint from the linkage.

APPLICATIONS

Large volume of 16-bit word data

If the main memory is too small to sort a large volume of one-word (or 16-bit word) data internally, the proposed block sort can be adopted to sort the large file externally merely by separating the one-word key into two components, say, key 1 and key 2. As the largest value of a one-word key is 32 767, the size of key 1 can be of 1–4 digits, counting from the left of the key value. This means that the original file may consist of as many as 3276 subfiles in order to reduce the sorting time.

Long key

For a 16-bit word computer, a long key of 5–9 digits (occupying a storage of 2 words) can be broken into two components to form a compound key (e.g. key 1–key 2). The proposed block sort is able to sort this compound key.

Generalized compound key

In principle, the proposed method can be extended to sort a generalized compound key (or called multi-component key) which may comprise key 1, key 2, key 3, etc. Based on the technique of the block sorting on a 2-component key, a key 1–cluster 1 pair (where cluster 1 contains the rest of components) can be established. After extracting key 2 from cluster 1, a key 2–cluster 2 pair is formed with respect to a key 2 address array. The same process is iterated until the last component is decomposed. In this repeated block sorting, records in nested clusters are then sorted and linked. Consequently, a sort file in a list structure is obtained.

CONCLUSION

The proposed block sort provides a method for sorting a large volume of records upon a two-component compound key externally, without applying a merging process and without destroying the physical location of the original disk file. Since the number of records in each subfile is practically small, a fast sort imposed on the subfile is insignificant in the global sort. The simple bubble sort is good enough to be adopted to sort each subfile in the proposed method. As the sort file is constructed in the list structure, insertion of a new record into the file or deletion of an inactive record from the file can be done easily. An explicit algorithm of the method, embedded with a bubble sort, is given in the Appendix. However, the bubble sort can be substituted by any other sorting method.

REFERENCES

1. D. E. Knuth, *The Art of Computer Programming*, Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts (1973).
2. F. K. Hwang and S. Lin, A simple algorithm for merging two disjoint linearly-ordered sets, *SIAM Journal of Computing* **1** (No. 1), (1972).
3. H. Lorin, *Sorting and Sort Systems*. Addison-Wesley, Reading, Massachusetts (1975).
4. C. R. Cook, and Do Jin Kim, Best sorting algorithm for nearly sorted lists, *Communications of the ACM* **23** (No. 11), 620 (November 1980).
5. R. C. Singleton, An efficient algorithm for sorting with minimal storage, *Communications of the ACM* **12** (March 1969).

Received February 1981

APPENDIX A

Subroutine of block sorting by a 2-component key

```
      SUBROUTINE COMPS(KTABL,ISIZE)
C     COMPS IS A SUBROUTINE OF A BLOCK SORT EMBEDDED WITH BUBBLE SORT OF
C     SMALL SUBFILES.
C     COMPS SORTS A  LARGE VOLUME OF RECORDS IN THE EXTERNAL STORAGE
C     AGAINST A COMPOUND KEY OF TWO COMPONENTS, NAMELY KEY1 AND
C     KEY2, IN ASCENDING ORDER. IT KEEPS THE ENTIRE FILE IN A LIST
C     STRUCTURE.
C     FOR SIMPLICITY, A RECORD CONTAINS ONLY KEY1, KEY2 AND THE POINTER.
C
C     DOCUMENTATION
C     COMPS---A SUBROUTINE OF COMPOUND-KEY SORT
C     KTABL---AN INDEX ARRAY WITH MAXIMUM SIZE EQUAL TO 9999 AND WITH
C          ---INDEX CORRESPONDING TO THE KEY1 VALUE
C     ISIZE---SIZE OF THE KEY1 VALUE
C     K2   ---AN ARRAY FOR KEY2
C     IDX  ---AN INDEX ARRAY ASSOCIATED WITH ARRAY K2
C     N    ---A RECORD NUMBER COUNTER
C          ---FINALLY,BEING THE TOTAL NUMBER OF RECORDS TO BE SORTED
C     KEY1 ---FIRST COMPONENT OF THE COMPOUND KEY
C     KEY2 ---SECOND COMPONENT OF THE COMPOUND KEY
C     IO   ---A POINTER INITIALIZED AS 0
C     IP   ---A POINTER POINTING TO THE NEXT RECORD
C     IHEAD---HEAD POINTER OF A SUBLIST
C     ITAIL---TAIL POINTER OF A SUBLIST
C     I,J  ---INDICES
C     IREC ---ITH RECORD
C     JREC ---JTH RECORD
C     IBND ---UPPER BOUND OF THE BUBBLE SORT
C     ITEMP---TEMPORARY STORAGE
C     LOC  ---TEMPORARY STORAGE
C     NEGHD---NEGATIVE HEAD POINTER, INDICATING THAT THERE IS ONLY ONE
C             RECORD IN THE LIST HAVING SUCH A KEY1 VALUE
C
      IMPLICIT INTEGER*2(I-N)
      DIMENSION KTABL(9999),K2(100),IDX(100)
      IO=0
      N=0
      DO 10 I=1,ISIZE
         KTABL(I)=0
   10 CONTINUE
C     TO SPLIT A DATA FILE INTO RANKED SUBFILES, BASED ON THE VALUE OF
C     KEY1.
  100 CONTINUE
      N=N+1
      READ (9,110,END=200) KEY1,KEY2
  110 FORMAT(2I5)
      IF (KTABL(KEY1)) 130,120,140
  120 KTABL(KEY1)=-N
      WRITE (1'N) KEY1,KEY2,IO
      GO TO 160
  130 IP=-KTABL(KEY1)
      GO TO 150
  140 IP=KTABL(KEY1)
  150 WRITE (1'N) KEY1,KEY2,IP
      KTABL(KEY1)=N
  160 GO TO 100
  200 CONTINUE
C     TO CONSTRUCT LINKED SUBFILES
      IHEAD=0
      ITAIL=0
      DO 420 I=1,ISIZE
         IF (KTABL(I)) 210,410,250
  210    IF (ITAIL) 220,220,230
  220    ITAIL=-KTABL(I)
         GO TO 240
  230    CONTINUE
         READ (1'ITAIL) KEY1,KEY2,IP
         WRITE (1'ITAIL) KEY1,KEY2,KTABL(I)
         ITAIL=-KTABL(I)
  240    CONTINUE
         GO TO 410
C     TO CONSTRUCT A KEY2 ARRAY FROM A SUBFILE
  250    IREC=KTABL(I)
         N=0
  260    CONTINUE
         N=N+1
         READ (1'IREC) KEY1,K2(N),IP
         IDX(N)=IREC
```

```
         IF (IP) 280,280,270
  270    IREC=IP
         GO TO 260
  280    CONTINUE
C     TO SORT KEY2 OF A SUBFILE BY THE BOBBLE SORT
C     (IT CAN BE SUBSTITUTED BY ANY OTHER SORTING METHOD)
         IBND=N-1
  300    CONTINUE
         LOC=0
         IF (IBND) 310,360,310
  310    DO 340 I=1,IBND
            IF (K2(I)-K2(I+1)) 330,330,320
  320       ITEMP=IDX(I)
            IDX(I)=IDX(I+1)
            IDX(I+1)=ITEMP
            ITEMP=K2(I)
            K2(I)=K2(I+1)
            K2(I+1)=ITEMP
            LOC=I
  330       CONTINUE
  340    CONTINUE
         IF (LOC) 350,360,350
  350    IBND=LOC-1
         GO TO 300
  360    CONTINUE
C     TO LINK RECORDS OF A SUBFILE IN ASCENDING ORDER
         NM1=N-1
         DO 380 J=1,NM1
            JREC=IDX(J)
            WRITE (1'JREC) KEY1,K2(J),IDX(J+1)
  380    CONTINUE
C     TO LINK THE SORTED SUBFILES INTO A COMPLETE SORT FILE
         KTABL(I)=IDX(1)
         IF (ITAIL) 400,400,390
  390    NEGHD=-IDX(1)
         READ (1'ITAIL) KEY1,KEY2,IP
         WRITE (1'ITAIL) KEY1,KEY2,NEGHD
  400    CONTINUE
         ITAIL=IDX(N)
  410    CONTINUE
  420 CONTINUE
      READ (1'ITAIL) KEY1,KEY2,IP
      WRITE (1'ITAIL) KEY1,KEY2,IO
      RETURN
      END
```