## Chapter 4.4 Two - Pass Algorithm Based On Sort

***Emphasis:***

For Two – Pass Algorithm, the data that come from Operating Object, are read into the main memory and dealt with some kinds of method and re-write to the disk, and re – read the disk and finish the Operation.

***Reasons Why in Two – Pass:***

1. Two – Pass Algorithm is enough even for the large Relation.
2. Multi – Pass Algorithm is not difficult after getting the Two – Pass Algorithm.

***Principle:***

Using the *Sort Operator T* to realize Two – Pass Algorithm. For Relation R that satisfies *B(R) > M*, divide them into M chunk and sort. After that sort the Sorted Chunk by using the method which makes the random chunk occupies only one memory block.

### Chapter 4.4.1 Two - Pass Multi - Way Merge Sort

***Concept:***

By using M memory buffer pool to sort, and sort to big Relation R by Two – Pass Algorithm, whose full name is Two – Pass Multi – Way Merge – Sort, TPMMS.

***Procedure:***

*Process 1:*

1. Put tuples from Relation R into M memory buffer pool continuously.
2. Use Main Memory Sort Algorithm to sort.
3. Store the Sorted Sub – table into the storage.

*Process 2:*

1. Merge the Sorted Sub – Tables.
2. At this stage, merge can work only on M – 1 Sorted Sub – Tables, which limits the size of Relation R.
3. Assign an input buffer block for each sub – table, and use a buffer block for output.

*(Usage of buffer area: The pointer which points to each buffer block presents the first element that has not been sorted and output.)*



***Merge Process:***

1. Find the smallest element among all Sub – Tables. Since comparison is finished in the main memory, therefore the linear search works.
2. Remove the first smallest element out of the block and put it into the output block.
3. The output block is full, then write the output block back into the disk and re – initialize the output block in order to store the next output block into the storage.
4. If the block which is used to remove the first smallest element is empty now, then read the next block of Sorted Sequence into the Buffer Pool;
5. If the next block is empty, then keep the input buffer pool empty, and do not take this block into consideration in the next Merge Process.

*(For the total M buffer blocks, the first Sorted Sequence could be as long as M length. Since there can only has M – 1 input buffer pool, so one block of Relation R could be as large as M \* (M -1), which is similar to M \* M.)*

***Cost:***

The total cost of Sort Operator is 3B but if the result needs to be stored to disk, then the total cost will be 4B.

***Example:***

Assume that the size of Block is 64k = 64 \* 1024, while 1G = 1024MB = 1024 \*1024 \* 1024 main memory.

1. M = 1024 \* 1024 \* 1024 / (64 \* 1024) = 16 \* 1024 = 16k. So one block of Relation R could be the size of 16 \* 1024 \* 16 \* 1024 = 2 ^ ( 4 + 10 + 4 + 10 ) = 2 ^ 28.
2. Since there can only has M – 1 input buffer pool, so the total size could be 2 ^ 28 \* 16k = 2 ^ 28 \* 16 \* 1024 = 2 ^ (28 + 4 + 10) = 2 ^ 42 or 4T size.

***Conclusion:***

TPMMS can be used to sort the big Relation, but if you have the bigger Relation, then you can use the Recursive Algorithm.

*(Divide the Relation into M \* (M – 1) piece, then use TPMMS algorithm to sort each piece and make the sorted piece as the result of the sub – table in the third round.)*

### Chapter 4.4.2 Remove Duplication Algorithm Through Sort

***Principle:***

Using the algorithm to remove duplication just like TPMMS sort tuples in Relation R.

***Process:***

1. In the second round, use the same method as TPMMS, assign one buffer block for each sorted sequence table as the input buffer block, and keep one output buffer block at the same time.
2. Copy the first element t of the block and neglect all other tuples which have the same values of tuple t.
3. Copy the tuple t into the output buffer block and delete all other t.
4. If output buffer block is full or empty, then deal with it just like in TPMMS.

***Cost:***

1. Disk I/O is *3B(R)*.
2. Using Remove Duplication Algorithm Through Sort only needs *B(R)^1/2*, but not B(R).

### Chapter 4.4.3 Grouping and Clustering Algorithm Based on Sort

***Procedure:***

1. Read tuples of Relation R into main memory with the size of M block. Using grouping attribute as sort key, then sort for every M blocks. Write the sequenced sub - table back to the disk.
2. Assign one main memory buffer to each sequenced sub - table and put the first block of each sequenced sub - table into the main memory buffer.
3. Search the tuple with the smallest tuple value from the first block in the main memory buffer. Record this tuple becomes the next group tuple:
4. Calculate the aggregation in this grouping list. *(Using the counting instead of getting average value.)*
5. Check whether the current tuple with the sorted key value equals to v, if equals to v, then count it to aggregation; Otherwise, neglect it.
6. If one block of the sequenced sub - table becomes empty, then just using the next block to substitute the current one. Do not stop until the sequenced sub - table turns to empty.

*(If there is no more other tuple with the sorted key value v, then output the current grouping attribute and its aggregation value, and formulate the final tuple.)*

***Cost:***

The total Disk I/O is *3B(R)*.

### Chapter 4.4.4 Union Algorithm Based on Sort

### Chapter 4.4.5 Intersection and Difference Based on Sort

### Chapter 4.4.6 One Simple Join Algorithm Based on Sort

### Chapter 4.4.7 Analysis of Simple Sort Join

### Chapter 4.4.8 One More Easy Join Algorithm Based on Sort

### Chapter 4.4.9 Conclusion about Algorithm Based on Sort