## 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 3 \* B but if the result needs to be stored to disk, then the total cost will be 4 \* B.

***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 *3 \* B(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 equals to *3 \* B(R)*.

### Chapter 4.4.4 Union Algorithm Based on Sort

***Principle:***

Only consider the algorithm for collection union.

***Procedure:***

1. In the first - pass, create Sorted Sequence Sub - Tables for Relation R and S.
2. Using one main memory buffer block for each Sorted Sequence Sub - Table, and use the first block of each Sorted Sequence Sub - Table to initialize the corresponding main memory buffer block.
3. Repeat finding the first tuple t from all main memory buffer blocks. Copy the tuple t to output, and delete all copies(equals to t) from the main memory buffer blocks. *(Using the same methods as the TPMMS when the input buffer block turns to empty or the output buffer block turns to full.)*

***Disk Cost:***

Read Relation R and S into main memory for twice, the first time is creating Sub - Tables for Relation R and S. The second time is sorting the Sub - Tables for Relation R and S. Also the Sorted Sequence Sub - Table need to be wrote back to the disk as one part of the newly created Sub - Table. The total disk I/O equals to *3 \* (B(R) + B(S))*.

***Buffer Cost:***

Each Sub - Table needs one main memory buffer block, and output also needs one buffer block. So the total number of Sub - Table doesn’t exceed M - 1. Therefore, the sum size of two Relation will not exceed M \* M, which means *B(R) + B(S) <= M \* M*.

### Chapter 4.4.5 Intersection and Difference Algorithm Based on Sort

***Principle:***

For Intersection and Difference Based Algorithm on Sort, just consider the smallest tuple t among all tuples in the input buffer block. Using the method below to generate the result and remove all copies of tuple t from input buffer block.

1. Collection Intersection Algorithm - If tuple t appears in both Relation R and S, then output tuple t.
2. Package Intersection Algorithm - The times of output tuple t is the smallest times that t appears in the Relation R and S. If one of the two count equals to 0, then do not output t, which means as long as tuple t doesn’t appear in the one or two relation, then give up to output the tuple t.
3. Collection Difference Algorithm - R - S, output tuple t as long as tuple t appears in Relation R but not Relation S.
4. Package Difference Algorithm - R - S, the times to output tuple t equals to the times tuple t appears in the Relation R minus the times tuple t appears in the Relation S. Of course, if tuple t appears in Relation S is far less than in Relation R, then give up output tuple t.

*(For Package Operation, when counting the times tuple t appears, there exists one situation that maybe all left tuples in one input buffer block equal to tuple t. Then if that happens, there may exists another situation that the next block of the sorted sequence sub - table still has more tuple t. So extra attention must be paid on it. This process may continue on several blocks or maybe on several sub - table.)*

***Cost:***

1. The total number of Disk I/O equals to *3 \* (B(R) + B(S))*.
2. In order to make the algorithm work, requires that *B(R) + B(S) <= M \* M*.

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

***Feature:***

When calculating join, two Relations on the connection property should have the same value. These same value tuples are put into the main memory, but the number may exceed the maximum value in the main memory.

***Improvement:***

In order to avoid this situation, we need to decrease the number of main memory used in other aspect but use them to save these same value tuples. In this chapter, we discuss one algorithm, using the smaller disk I/O, based on Sort, but this algorithm may has some problem when lots of tuples have the same value in the connection attribute.

***Procedure:***

Known Relation R(X, Y) and S(Y, Z), and using M main memory blocks as the buffer block:

1. Use Y attribute as the sorted key, and use the TPMMS to sort Relation R.
2. Sort Relation S as the same way as Relation R.
3. Merge the sorted Relation R and Relation S. Using two buffer blocks, one is for the current block of Relation R while another is for the current block of Relation S.
4. Find the smallest value y of Join Attribute Y in the front block of Relation R and Relation S.
5. If value y doesn’t exist in the front block of another Relation, then just delete the tuple that contains value y in the current Relation.
6. Otherwise, find all tuples that contain the sorted key value y in two Relations. If need, just read next block from sorted Relation R and Relation S, until make sure that there is no more copies of value y in both Relations. Using M buffer blocks at most.
7. Output all join tuples with the same sorted key value y in two Relation R and Relation S.
8. If there is no more other tuples in one Relation that need to be considered, then re-load buffer block setting for the Relation.

***Example and Cost:***

For Relation R and S, Relation R occupies 1000 blocks and Relation S occupies 500 blocks, and M = 101. When use TPMMS on one Relation, then for each block, we use 4 disk I/O, two times for each phase. So, we need to use 4 \* (B(R) + B(S)) disk I/O to sort Relation R and Relation S or 6000 disk I/O.

During the Merge Phase, get the connected tuples, using another 1500 disk I/O to read each block of Relation R and S. In the Merge, only two buffer blocks are needed. If needs, then we can use all 101 blocks to contain all tuples that have the join property value y.

*The total amount of disk I/O equals to 7500, but in the Nested Loop Join, it only equals to 5500. But the Nested Loop Join is a Quadratic Algorithm, the time is proportion to B(R) \* B(S), while Sort Join is a Liner Algorithm, the time is proportion to B(R) + B(S).*

Normally use the Sort Join, but because of the constant factor and comparatively small Relation, then we choose the Nested Loop Join Algorithm.

### Chapter 4.4.7 Analysis of Simple Sort Join

***Instruction:***

The main restriction is that the Merge Sort can be used on the Relation R and Relation S. In order to ensure that TPMMS can be used on Relation R and Relation S, then we need to make sure that *B(R) <= M \* M*, *B(S) <= M \* M*. Also, we need to make sure that all tuples with the Join Attribute value y should all can be placed into M main memory.

1. Simple Sort Join uses *5 \* (B(R) + B(S))* disk I/O.
2. In order to work, it requires that *B(R) <= M \* M* and *B(S) <= M \* M*.
3. It also requires that all tuples with the attribute value used for connection can be placed into M main memory.

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

***Definition:***

If there have not so many tuples with the same values on Sorted Key Attribute, then just combine Merge process with Join process, then for each block, we can save two disk I/O. We call this process *Sort - Join*; The algorithm *Merge - Join* and *Sort - Merge - Join* is also this algorithm.

***Procedure:***

Using M buffer blocks to calculate R(X, Y) Join S(Y, Z), then:

1. Using Y as the Sorted Key, create M Sorted Sequenced Sub - Table for Relation R and Relation S.
2. Call first block of each Sub - Table into buffer block; Assume that the total Sub - Table will not exceed M.
3. Find the smallest value y of Attribute Y from all Sub - Table. Identify all tuples with value y, if the number of Sub - Table is smaller than M, then use one part of M buffer memory to contain these tuples. Output the connection of tuples with the same value y of Relation R and Relation S. If one buffer block for one Sub - Table is used up, then relocate the block into the buffer block.

***Example:***

* Relation R and Relation S, use 101 buffer blocks to join R and S, they have 1000 blocks and 500 blocks. Then divide Relation R into 10 Sub - Tables and Relation S into 5 Sub - Tables, each with 100 length, and sort them.
* Then use 15 blocks to contain the current block of 15 Sub - Tables. If many tuples have the fixed y value, then use the left 86 buffer blocks to store these tuples.
* For each data block to execute three disk I/O. Then two times is used to create sorted Sub - Tables. Then in the Merge process, each block of each Sequenced Sub - Table needs to be read into main memory, therefore the total disk I/O is 4500.

***Cost:***

1. The total disk I/O equals to *3 \* (B(R) + B(S))*.
2. The length of Sub - Table is M, and the total Sub - Table is at most M. The total Sub - Table of two Relation is at most M. *B(R) + B(S) <= M \* M*.

### Chapter 4.4.9 Conclusion on Algorithm Based on Sort

|  |  |  |
| --- | --- | --- |
| Operator | M | Disk I/O |
| Grouping | B ^ 1/2 | 3 \* B |
| Sorting | B ^ 1/2 | 3 \* B |
| Remove Duplication | B ^ 1/2 | 3 \* B |
| Union | (B(R) + B(S)) ^ 1/2 | 3 \* (B(R) + B(S)) |
| Intersection | (B(R) + B(S)) ^ 1/2 | 3 \* (B(R) + B(S)) |
| Difference | (B(R) + B(S)) ^ 1/2 | 3 \* (B(R) + B(S)) |
| Join | (max(B(R), B(S))) ^ 1/2 | 5 \* (B(R) + B(S)) |
| Join | (B(R) + B(S)) ^ 1/2 | 3 \* (B(R) + B(S)) |