Query Processing 3: Sort Join

Yufei Tao

https://www.cse.cuhk.edu.hk/~taoyf

This lecture will introduce the **sort join** algorithm for computing a natural join involving two relations.

The Binary Join Problem

$R_1(X, Y)$: A relation with attributes X and Y.
$R_2(X, Z)$: A relation with attributes X and Z.
B_1 = the number of disk blocks that R_1 occupies.
B_2 = the number of disk blocks that R_2 occupies.
M = the number of memory blocks (a.k.a., the buffer blocks).

- We assume $M \geq 3$.

Goal: Compute the join result $R_1 \bowtie R_2$.

We will carry out our discussion under the following assumption:

> **No skew assumption:**
> For any X-value, the tuples of R_1 having that X-value fit in at most $M - 2$ blocks.

Sort Join

Step 1: Sort $R_1(X, Y)$ on X, and sort $R_2(X, Z)$ on X

- Using the external sort algorithm.

Step 2: Scan the sorted R_1 and R_2 synchronously to output the result.

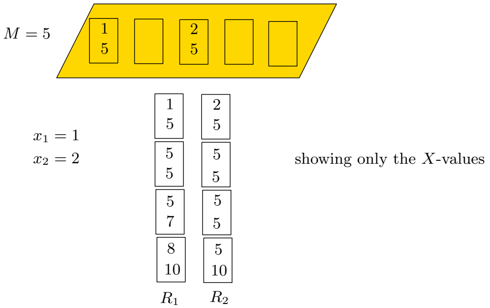
Next, we will explain how to do Step 2 in $B_1 + B_2$ I/Os.

Sort Join: Step 2

We will maintain a value x_1 for R_1 and a value x_2 for R_2 to enforce the following invariant:

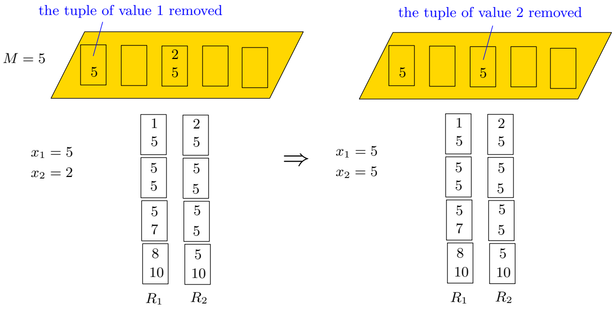
> **Invariant:** All the result tuples of R_1 (resp., R_2) with X-values less than x_1 (resp., x_2) have been output.

In the beginning, load the first blocks of R_1 and R_2 into memory.
Set x_1 (resp., x_2) to the X-value of the first tuple of R_1 (resp., R_2).



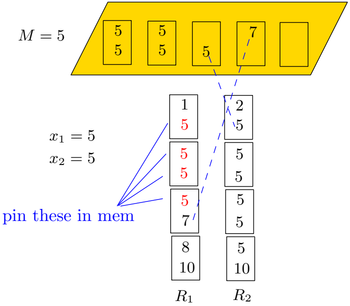
Sort Join: Step 2

If $x_1 < x_2$, move x_1 to the next element of R_1 (loading the next block of R_1 into memory if necessary). **If** $x_2 < x_1$, move x_2 to the next element of R_2 (loading the next block of R_2 into memory if necessary).



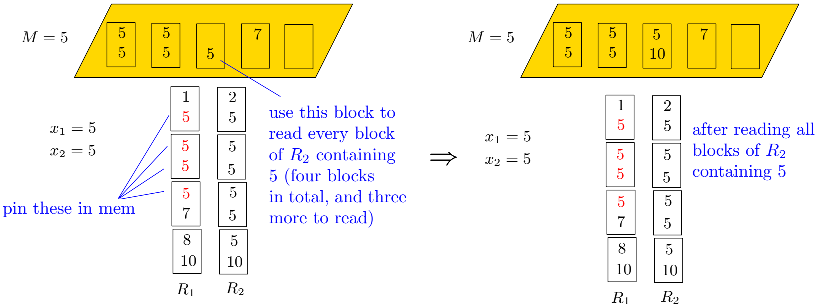
Sort Join: Step 2

If $x_1 = x_2$, read into memory all tuples of R_1 with X-values equal to x_1. By the no-skew assumption, they occupy at most $M - 2$ memory blocks.



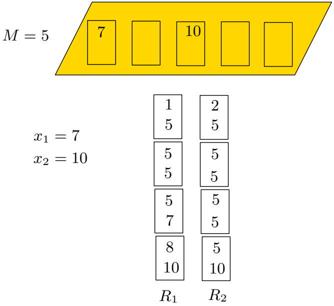
Sort Join: Step 2

Then use one memory block to read every block of R_2 containing tuples whose X-values equal x_2 (one block at a time). For each block read, produce all the join result tuples whose X-values equal x_2.



Sort Join: Step 2

Move x_1 to the next element of R_1 (reading the next block of R_1 if necessary). Move x_2 to the next element of R_2 (reading the next block of R_2 if necessary).



The algorithm continues in the same fashion until R_1 or R_2 is exhausted.

Every block of R_1 and R_2 is read exactly once. Hence, Step 2 performs $B_1 + B_2$ I/Os.

The total I/O cost of the sort join algorithm is bounded by $sort(B_1) + sort(B_2) + B_1 + B_2$ where $sort(x)$ is the I/O cost of sorting x blocks of tuples.

In practice, we typically have $B_1 \le M(M-1)$ and $B_2 \le M(M-1)$, in which case the total I/O cost is bounded by $5(B_1 + B_2)$.

Remark: The above analysis has not considered the cost of writing the join result to the disk. If that is necessary, you should add B_{out} to the I/O cost, where B_{out} is the number of blocks needed to store the join result, provided that you have an additional memory block to serve as the output buffer.

Here is a question for you:

The 3-Star Join Problem

$R_1(X, A_1)$: A relation with attributes X and A_1.
$R_2(X, A_2)$: A relation with attributes X and A_2.
$R_3(X, A_3)$: A relation with attributes X and A_3.
B_i = the number of disk blocks that R_i occupies ($1 \le i \le 3$).
M = the number of memory blocks.

If you are to modify the sort-join algorithm to compute

$$R_1 \bowtie R_2 \bowtie R_3$$

in $5(B_1 + B_2 + B_3)$ I/Os, what assumptions would you need?

Returning to the binary join problem $R_1(X, Y) \bowtie R_2(X, Z)$, next we outline a refinement of the sort join algorithm that can reduce the I/O cost to $3(B_1 + B_2)$ when the memory size (i.e., M) is reasonably large.

Remark: This content will not be tested.

Refined Sort Join

Perform the **initial step** of external sort on R_1 (sorting attribute $= X$).
Perform the **initial step** of external sort on R_2 (sorting attribute $= X$).
The above needs $2(B_1 + B_2)$ I/Os and yields

- $n_1 = \lceil B_1/M \rceil$ **sorted runs** of R_1, each having M blocks;
- $n_2 = \lceil B_2/M \rceil$ **sorted runs** of R_2, each having M blocks.

We assume

- $n_1 + n_2 < M$;
- (**skew condition modified**) for any X-value, the tuples of R_1 having that X-value fit in at most $M - n_1 - n_2$ blocks.

Under these assumptions, it is possible to scan all the sorted runs synchronously to output $R_1 \bowtie R_2$ directly in $B_1 + B_2$ I/Os (**hint**: how did you solve the 3-star join problem?).