COL 362 & COL 632

Analysis of Sorting, Joins

21 Mar 2023

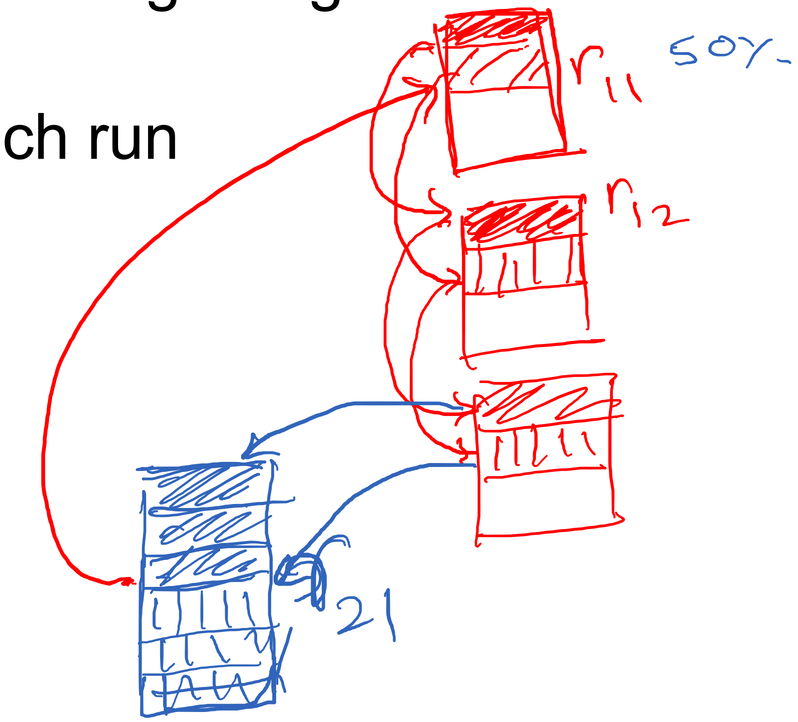
Reducing Seeks in External Sort

b_r blocks

b_b blocks per run, (buffer)

- 1 block per run leads to too many seeks during merge
- Instead use b_b buffer blocks per run
  => read/write b_b blocks at a time from each run
- Can merge $\left\lfloor \frac{M}{b_b} \right\rfloor - 1$ runs in one pass
- Total number of merge passes required:

$$\left\lceil \log_{\left\lfloor \frac{M}{b_b} \right\rfloor - 1} \frac{b_r}{M} \right\rceil$$



External Sort Merge – Block Transfers

- Block transfers for initial run creation as well as in each pass is $\underline{2b_r}$
  - For the final pass, we don't count write cost
  - We can ignore final write cost for all operations since the output of an operation may be sent to the parent operation without being written to disk

when can we ignore the final pass.?

pipelining

- Thus, total number of **block transfers** for external sorting:

$$\boxed{b_r \Big(2\left\lceil\log_{\left\lfloor \frac{M}{b_b}\right\rfloor-1}\frac{b_r}{M}\right\rceil}+1\Big)$$

$$2b_r\left\lceil\log_{\left\lfloor \frac{M}{b_b}\right\rfloor-1}\frac{b_r}{M}\right\rceil + \underline{b_r}$$

External Merge Sort - Cost of seeks

- **During run generation**: one seek to read each run and one seek to write each run
  - $2\left\lceil \frac{b_r}{M} \right\rceil$
- **During the merge phase:**
  - Need $\boxed{2\left\lceil \frac{b_r}{b_b} \right\rceil}$ seeks for each merge pass
  - except the final one which does not require a write
- **Total number of seeks**:

$$2\left\lceil \frac{b_r}{M} \right\rceil + \underbrace{\left\lceil \frac{b_r}{b_b} \right\rceil \left(2\left\lceil \log_{\left\lfloor \frac{M}{b_b} \right\rfloor - 1} \frac{b_r}{M} \right\rceil \cancel{-1}\right)}$$

Ramakrishna, Gehrke . – Cow Book

Number of Passes (Tentative)

N	$b_b = 3$	$b_b = 5$	$b_b = 9$	$b_b = 17$	$b_b = 129$	$b_b = 257$
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Up-to-date sort performance at http://sortbenchmark.org

Join Operation

- Several different algorithms to implement joins physical plans
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join extn. from external sort merge
  - Hash-join
- Choice based on cost estimate
- Examples use the following information
  - Number of records of *student*: 5,000 *takes*: 10,000
  - Number of blocks of *student*: 100 *takes*: 400

Nested-Loop Join

- To compute the theta join $r \bowtie_{\theta} s$

```
    for each tuple $t_r$ in r do begin          outer loop
        for each tuple $t_s$ in s do begin      inner loop
         test pair $(t_r, ts)$ to see if they satisfy the join condition $\theta$
         if they do, add $t_r \bullet t_s$ to the result.
        end
    end
```

- *r* is called the **outer relation** and *s* the **inner relation** of the join.
- Requires no indices and can be used with any kind of join condition.
- Expensive since it examines every pair of tuples in the two relations.

Nested-Loop Join (Cont.)

- In the **worst case**, if there is enough memory only to hold one block of each relation, the estimated cost is
  $\underline{\boxed{n_r * \boxed{bs}} + b_r}$ block transfers, plus $\underline{n_r + b_r}$ seeks
- Assuming worst case memory availability cost estimate is
  - with *student* as outer relation:
    - 5000 $*$ 400 + 100 = 2,000,100 block transfers,
    - 5000 + 100 = 5100 seeks
  - with *takes* as the outer relation
    - 10000 $*$ 100 + 400 = 1,000,400 block transfers and 10,400 seeks
- If the smaller relation fits entirely in memory, use that as the inner relation
  - Reduces cost to $b_r + bs$ block transfers and 2 seeks

Block Nested-Loop Join

- Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.

```
for each block $B_r$ of $r$ do begin
    for each block $B_s$ of $s$ do begin
        for each tuple $t_r$ in $B_r$ do begin
            for each tuple $t_s$ in $B_s$ do begin
                Check if $(t_r, t_s)$ satisfy the join condition
                if they do, add $t_r \cdot t_s$ to the result.
            end
        end
    end
end
```

Block Nested-Loop Join (Cont.)

- Worst case estimate: $\boxed{b_r * b_s} + b_r$ block transfers plus $\boxed{2 * b_r}$ seeks
  - Each block in the inner relation *s* is read once for each *block* in the outer relation
- Best case ?
- In block nested-loop, use $M - 2$ disk blocks as blocking unit for outer relations, where *M* = memory size in blocks; use remaining two blocks to buffer inner relation and output
  - Cost = $\left\lceil \frac{b_r}{M-2} \right\rceil \times b_s + b_r$ block transfers + 2 $\left\lceil \frac{b_r}{M-2} \right\rceil$ seeks
- If equi-join attribute forms a key or inner relation, stop inner loop on first match
- Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)