University of Edinburgh
School of Informatics
INFR11199 - Advanced Database Systems (Spring 2024)

Tutorial Sheet 3

1. (Sorting and Hashing) Suppose the size of a page is 4 KB, and the size of the memory buffer is 1 MB (1024 KB).

   (a) We have a relation of size 800 KB. How many page I/Os are required to sort this relation and write the sorted relation back to disk?

   > **Solution:** 400 (= 200 + 200).
   > 200 to read in, 200 to write out. Since the relation is small enough to completely fit into the buffer, we only need to read it in, sort it (no I/Os required for sorting), then write the sorted pages back to disk.

   (b) We have a relation of size 5000 KB. How many page I/Os are required to sort this relation and write the sorted relation back to disk?

   > **Solution:** 5000 (= 2 * 1250 * number of passes).
   > 2 passes. 5000 KB with 4KB per page means 1250 pages are needed to store the relation. We have B = 1024 / 4 = 256 pages in our buffer.
   > Number of Passes = $1 + \lceil log_{255} \lceil 1250/256 \rceil \rceil = 2$.

   (c) What is the size of the largest relation that would need two passes to sort?

   > **Solution:** 261,120 KB. (255 * 256 pages).

   (d) What is the size of the largest relation we can possibly hash in two passes (i.e., with just one partitioning phase)?

   > **Solution:** 261,120 KB.

(e) Suppose we have a relation of size 3000 KB. We are executing a `DISTINCT` query on a column `age`, which has only two distinct values, evenly distributed. Would sorting or hashing be better here, and why?

> **Solution:** Hashing, which allows us to remove duplicates early on and potentially improve performance (in this case, we might be able to finish in 1 pass, instead of 2 for sorting).

(f) Now suppose we were executing a `GROUP BY` on `age` instead. Would sorting or hashing be better here, and why?

> **Solution:** Sorting because hashing won't work; each partition is larger than memory, so no amount of hash partitioning will suffice.

2. (Joins) Consider the following database of students and assignment submissions and the SQL query:

```sql
CREATE TABLE Students (
  student_id INTEGER PRIMARY KEY,
  ...
);
CREATE TABLE Assignments(
  assignment_number INTEGER,
  student_id INTEGER REFERENCES Students(student_id),
  ...
);
SELECT *
  FROM Students, Assignments
 WHERE Students.student_id = Assignments.student_id;
```

Assume the following:

- `Students` has 20 pages, with 200 records per page
- `Assignments` has 40 pages, with 250 records per page.

(a) What is the I/O cost of a simple nested loop join for `Students` ⋈ `Assignments`?

> **Solution:** 160,020 I/Os.
> The cost of a SNLJ is: #pages(S) + #records(S) · #pages(A).
> Plugging in the numbers gives us $20 + (20 \cdot 200) \cdot 40 = 160,020$ I/Os.

(b) What is the I/O cost of a simple nested loop join for `Assignments` ⋈ `Students`?

Solution: 200,040 I/Os.
The cost of a SNLJ is: #pages(A) + #records(A) · #pages(S).
Plugging in the numbers gives us $40 + (40 \cdot 250) \cdot 20 = 200,040$ I/Os.

(c) What is the I/O cost of a block nested loop join for `Students` ⋈ `Assignments`? Assume our buffer size is $B = 12$ pages.

Solution: 100 I/Os.
Since `Students` is the outer table, we calculate the number of blocks of `Students`: #pages(S) / (B-2) = 20 / 10 = 2. Thus, the final cost is #pages(S) plus 2 passes through all pages(A), or $20 + 2 \cdot 40 = 100$ I/Os.

(d) What is the I/O cost of a block nested loop join for `Assignments` ⋈ `Students`? Assume our buffer size is $B = 12$ pages.

Solution: 120 I/Os.
Since `Assignments` is the outer table, we calculate the number of blocks of `Assignments`: #pages(A) / (B-2) = 40 / 10 = 4. Thus, the final cost is #pages(A) plus 4 passes through all pages(S), or $40 + 4 \cdot 20 = 120$ I/Os.

(e) What is the I/O cost of an Index-Nested Loop Join for `Students` on ⋈ `Assignments`?

Assume we have a *clustered* variant B index on `Assignments.student_id`, in the form of a height 2 B+ tree. Assume that: index (non-leaf) nodes and leaf pages are not cached; all hits are on the same leaf page; and all hits are also on the same data page.

Solution: 16,020 I/Os.
The formula is #pages(S) + #records(S) * (cost of index lookup).
The cost of index lookup is 3 I/Os to access the leaf, and 1 I/O to access the data page for all matching records.
So the total cost is $20 + 4000 \cdot 4 = 16,020$ I/Os.

(f) Now assume we have an *unclustered* variant B index on `Assignments.student_id`, in the form of a height 2 B+ tree. Assume that index node pages and leaf pages are never cached, and we only need to read the relevant leaf page once for each record of `Students`, and all hits are on the same leaf page.

What is the I/O cost of an Index-Nested Loop Join for `Students` ⋈ `Assignments`?

Hint: The foreign key in `Assignments` may play a role in how many accesses we do per record.

Solution: 22,020 I/Os.
The formula is #pages(S) + #records(S) * (cost of index lookup).
This time though, the cost of index lookup is 3 I/Os to access the leaf, and 1 I/O to access the data page for *each matching record*.
How many records match per key? We actually haven't told you! But, we do know that we will eventually have to access each record exactly once (since each `Assignments` is foreign-keyed on a `student_id` – so there will be #records(A) = 10,000 data page lookups, one for each row.
So the total cost is $20 + 4000 \cdot 3 + 10000 = 22,020$ I/Os.

(g) What is the cost of an *unoptimized* sort-merge join for `Students` $\bowtie$ `Assignments`? Assume we have $B = 12$ buffer pages.

Solution: 300 I/Os.
The formula is (cost of sorting S) + (cost of sorting A) + #pages(S) + #pages(A).
For sorting S: The first pass will make two runs, which is mergeable in one merge pass; thus, we need two passes.
For sorting A: The first pass will make four runs, which is mergeable in one merge pass; thus, we need two passes.
Thus the total cost is $(2 \cdot 2\#pages(S)) + (2 \cdot 2\#pages(A)) + \#pages(S) + \#pages(A) = 5(\#pages(S) + \#pages(A)) = 5 \cdot 60 = 300$ I/Os.

(h) What is the cost of an *optimized* sort-merge join for `Students` $\bowtie$ `Assignments`? Assume we have $B = 12$ buffer pages.

Solution: 180 I/Os.
The difference from the above question is that we will skip the last write in the external sorting phase, and the initial read in the sort-merge phase. For this to be possible, all the runs of S and A in the last phase of external sorting should be able to fit into memory together. From the previous question, we know there are 2 + 4 = 6 runs, which fits just fine in our buffer of 12 pages.
The total cost is $300 - 2\#pages(S) - 2\#pages(A) = 300 - 120 = 180$ I/Os.

(i) In the previous question, we had a buffer of $B = 12$ pages. If we shrank B enough, the answer we got might change.
How small can the buffer B be without changing the I/O cost answer we got?

Solution: The restriction for optimized sort-merge join is that the number of final runs of S and A can both fit in memory simultaneously (i.e., the

number of runs of S + the number of runs of $A \leq B - 1$). We had 2 + 4 runs last time, which fit comfortably in $12 - 1$ buffer pages (recall that one page is reserved for output).
What about $B = 11$? We would still have $2 + 4 \leq 11 - 1$ runs.
What about $B = 10$? We would still have $2 + 4 \leq 10 - 1$ runs.
What about $B = 9$? Now we have 3 runs for S and 5 runs for A, which just exactly fits in $9 - 1$ buffer pages.
Since 9 buffer pages fits perfectly, any smaller would force more merge passes and thus more I/Os.

(j) What is the I/O cost of Grace Hash Join on these tables?
Assume uniform hash partitioning and a buffer pool consisting of $B = 6$ pages.

Solution: 180 I/Os.
For Grace Hash Join, we have to walk through what the partition sizes are like for each phase, one phase at a time. In the partitioning phase, we will proceed as in external hashing. We will load in 1 page a time and hash it into $B - 1 = 5$ partitions. This means the 20 pages of S get split into 4 pages per partition, and the 40 pages of A get split into 8 pages per partition.

Do we need to recursively partition? No! Remember that the stopping condition is that any table's partition fits in $B - 2 = 4$ buffer pages; the partitions of S satisfy this.

In the hash joining phase, the I/O cost is simply the total number of pages across all partitions – we read all of these in exactly once.

Thus the final I/O cost is 20 + 20 for partitioning S, 40 + 40 for partitioning A, and 20 + 40 for the hash join, for a total cost of 180 I/Os.