INFO20003 Tutorial – Week 7 Solutions

(Tutorial: Query processing and cost estimation)

Objectives:

This tutorial will cover:

  I.   Effect of index on selection operator – 10 mins
 II.   Matching index – 10 mins
III.   Cost estimation for different joins – 30 mins

Exercises:

1. Question about the effect of index on selection:

Consider a relation R (a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that R.a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R.a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file of R directly.
- Use a B+ tree index on attribute R.a.
- Use a hash index on attribute R.a.

Queries:

a. $\sigma_{a < 50000}$ (R)
The sorted file over R is preferred in this case – we just start from the beginning producing results directly.

b. $\sigma_{a = 50000}$ (R)
This is an equality query. A hash index will be the most cost-effective option in this case.

c. $\sigma_{a > 50000 \land a < 50010}$ (R)
This is a range query which does not begin at the start of the sorted file. A B+ tree index should be the cheapest of all the options.

2. Matching index

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given selection conditions and briefly explain why.

- A B+ tree index on the search key (Sailors.sid)

    a. $\sigma_{Sailors.sid < 50,000}$ (Sailors)
        Match, primary conjuncts are: Sailors.sid < 50,000

b. $\sigma_{\text{Sailors.sid} = 50,000}$ (Sailors)

  Match, primary conjuncts are: Sailors.sid = 50,000

- A hash index on the search key (Sailors.sid)
    c. $\sigma_{\text{Sailors.sid} < 50,000}$ (Sailors)

      No match, range queries cannot be applied to a hash index.

    d. $\sigma_{\text{Sailors.sid} = 50,000}$ (Sailors)

      Match, primary conjuncts are: Sailors.sid = 50,000

- A B+ tree index on the search key (Sailors.rating, Sailors.age)
    e. $\sigma_{\text{Sailors.rating} < 8 \,\wedge\, \text{Sailors.age} = 21}$ (Sailors)

      Match, primary conjuncts are Sailors.rating < 8 and
      Sailors.rating < 8 ∧ Sailors.age = 21

    f. $\sigma_{\text{Sailors.rating} = 8}$ (Sailors)

      Match, primary conjuncts are: Sailors.rating = 8

    g. $\sigma_{\text{Sailors.age} = 21}$ (Sailors)

      No match. The index on (Sailors.rating, Sailors.age) is primarily sorted on Sailors.rating, so the entire relation would need to be searched to find those sailors with a particular Sailors.age value.

3. **Question about the cost analysis of different joins:**

Consider the join R $\bowtie_{\text{R.a = S.b}}$ S, given the following information about the relations to be joined:

- Relation R contains 10,000 tuples and has 10 tuples/page.
- Relation S contains 2,000 tuples and also has 10 tuples/page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

The cost metric is the number of page I/Os unless otherwise noted and the cost of writing out the result should be uniformly ignored.

Let $M = \frac{10,000}{10} = 1000$ be the number of pages in R, $N = \frac{2,000}{10} = 200$ be the number of pages in S, and $B = 52$ be the number of buffer pages available.

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

  The basic idea of page-oriented nested loops join is to do a page by page scan of the outer relation and for each outer page, do a page-by-page scan of the inner relation. The cost of joining R and S can be minimised by selecting the smaller relation as the outer relation. We will select S as the outer relation and compute cost as follows:

  Total cost = (# of pages in outer) + (# of pages in outer × # of pages in inner)
  $$= N + (N \times M) = 200 + (200 \times 1000) = 200,200$$

In this algorithm we don't use multiple buffers at a time, so the minimum requirement is one input buffer to page through each relation and one output buffer to store the output. Hence in total 3 buffer pages are required.

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

In block nested loops join, the outer relation is read in blocks (groups of pages that will fit into whatever buffer pages are available), and, for each block, do a page-by-page scan of the inner relation. The outer relation is scanned once, and the inner relation is scanned only once for the outer block. Assuming there are B buffers available,

$$\text{\# of blocks} = \text{ceil}\left(\frac{\text{\# of pages in outer}}{B-2}\right) = \text{ceil}\left(\frac{200}{50}\right) = 4$$

Total cost = (# of pages in outer) + (# of blocks × # of pages in inner)
                  = 200 + (4 × 1000) = 4200

If we have fewer buffers available, the cost will increase as the # of blocks will vary. The minimum number of buffer pages is 52 for this cost.

c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.

We will use external merge sort for this case. The initial sorting pass will split R into 20 runs of 50 buffer pages. After that, these pages will be written to the disk. These sorted pages will be read again to be compared across runs and combine the results. The resulting sorted pages will be written to disk. Similarly, S will split into 4 runs of approximately 50 buffer pages and follow the same process as R. The general formula for external merge sort is $2N \times$ # of passes. As we only require two passes to sort R and S, the cost analysis is as follows:

Cost of sorting R = 2 × # of passes × # of pages of R
                           = 2 × 2 × 1000 = 4000

Cost of sorting S = 2 × 2 × 200 = 800

Cost of merging R and S = # of pages read of R + # of pages read of S
                                        = 1000 + 200 = 1200

Total cost = Cost of sorting R + Cost of sorting S + Cost of merging R and S
                  = 4000 + 800 + 1200 = 6000

d. What is the cost of joining R and S using the **Hash Join** algorithm?

In hash join, each relation is partitioned and then the join is performed by "matching" elements from corresponding partitions.

Total cost = $3(M + N)$
                  = 3(1000 + 200) = 3600

e. What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

The optimal cost would be achieved if each relation was read only once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page by page and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation and one page to serve as an output buffer.

Total cost $= M + N = 1200$

The minimum number of buffer pages for this cost is min$\{M, N\} + 1 + 1 = 202$.