COMP0178: Database Fundamentals (Term 1)

Query processing worksheet – solutions

For the following questions, express your answers in terms of the following statistics:

nTuples(R) – cardinality of R
bFactor(R) – blocking factor of R
nBlocks(R) – number of blocks in R
$nDistinct_A(R)$ – number of distinct values for A in R
$min_A(R)$, $max_A(R)$, – range endpoints for A in R
$SC_A(R)$ – selection cardinality of A in R

1. Estimate the size of the result returned by a selection under the following conditions:
    a. Inequality conditional (A > c)
    $((max_A(R) – c) / (max_A(R) – min_A(R)) * nTuples(R)$ *(portion of range that fulfills condition, applied to number of tuples)*

    b. Equal to any element in set $\{c_1, ..., c_n\}$
    (n / nDistinct(R)) * nTuples(R) *(portion of values covered by this set, applied to number of tuples)*

2. Estimate the cost in disk accesses for a selection under the following conditions:
    a. Linear search on unordered file with no index
        i. Equality condition on unique attribute
        nBlocks(R) *in worst case,* **nBlocks(R)/2** *in average case*

        ii. Equality condition on non-unique attribute or inequality condition –
        nBlocks(R) always

    b. Binary search on ordered file with no index
        i. Equality condition on unique attribute
        lg(nBlocks(R))

        ii. Equality condition on non-unique attribute
        $lg(nBlocks(R)) + SC_A(R)/bFactor(R) – 1$

    c. Hash lookup on hashed attribute
    generally 1

    d. Index lookup on indexed attribute
    1, or 1 + $nLevels_A(I)$ *(number of levels in multi-level index I on attribute A)*

3. Estimate the size of the result returned by a join under the following conditions:
    a. R equijoin S on a key in R
    nTuples(S) *(at most one match for each tuple in S)*

b. R join S on some non-unique attribute
SC_A(R) * nTuples(S), *or* **SC_A(S) * nTuples(R)** *(expect SC_A(R) tuples to match each tuple in S and vice versa)*

4. Estimate the cost in disk accesses for a join under the following conditions:
    a. Basic nested loop join
        i. When all blocks for R and S can be loaded into memory
        nBlocks(R) + nBlocks(S) *(load each block exactly once and do all processing in memory)*

        ii. Loading one block of R (outer loop) and loading each block of S (inner loop) one at a time
        nBlocks(R) + nBlocks(R) * nBlocks(S) *(load each block of R exactly once, load each block of S nBlocks(R) times)*

        iii. Loading (maxBlocks – 2) blocks of R at a time and joining with one block of S (inner loop) at a time
        nBlocks(R) + (nBlocks(R)/(maxBlocks-2)) * nBlocks(S) *(load each block of S however many times it takes to load each block of R once, loading maxBlocks – 2 blocks at a time)*

    b. Sort-merge join
        i. When the relations are already sorted on the join attribute
        nBlocks(R) + nBlocks(S)

        ii. When the relations are not sorted on the join attribute
        nBlocks(R) + nBlocks(S) + nBlocks(R) * lg(nBlocks(R)) + nBlocks(S) * lg(nBlocks(S)) *(add cost of sorting which is n log n)*

5. Give upper and lower bounds for the size of the results of a:
    a. Set union
    lower: **max(nTuples(R), nTuples(S))**
    upper: **nTuples(R) + nTuples(S)**

    b. Set intersection
    lower: **0**
    upper: **min(nTuples(R), nTuples(S))**

    c. Set difference
    lower: **0**
    upper: **nTuples(R)**