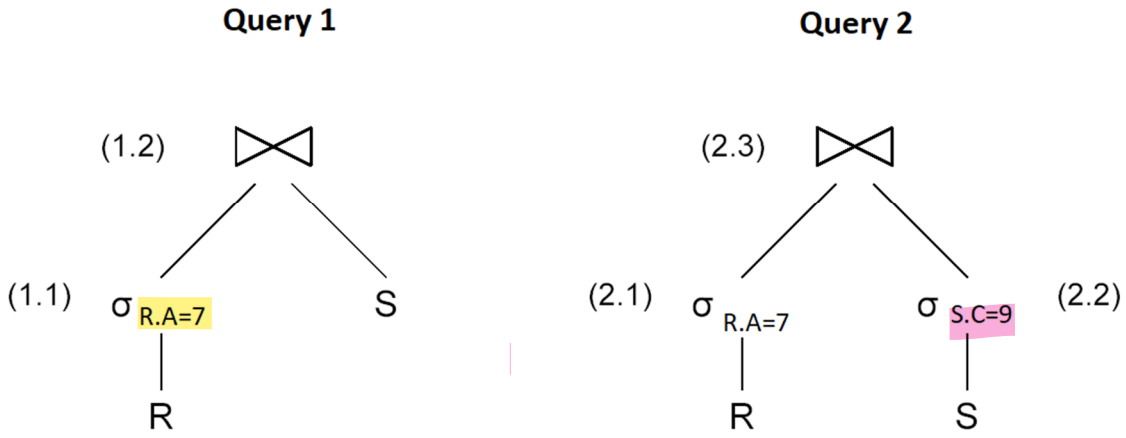
Implementation of DBMS
Exercise Sheet 13, Solutions
Klingemann, WS 2023 / 2024

1) Consider the following queries on relations R(A, B) and S(B, C):
Query 1: SELECT * FROM R, S WHERE R.B = S.B AND R.A = 7
Query 2: SELECT * FROM R, S WHERE R.B = S.B AND R.A = 7 AND S.C=9
The following two plans are being considered:



Relation R has 60,000 tuples with 10 tuples per disk block, V(R, A) = 6 and V(R, B) = 12. Similarly, S has 30,000 tuples with 30 tuples per disk block, V(S, B) = 5 and V(S, C) = 20. Assume values are distributed over possible V(Relation, Attribute) values (not over possible domain values).
In this exercise we make the following additional assumptions:

- The join is implemented as a hash-join (without optimization);
- The intermediate result produced by operation (1.1) is not written to disk;
- Hash buckets are stored on disk;
- The final result is kept in main-memory;
- There is enough memory to execute the hash join algorithm.

a) How many disk I/O's does query 1 require?
b) Also assume that the result of neither (2.1) nor (2.2) is stored to disk.
How many disk I/O's does query 2 require?
Solution: We have
B(R) = 60000 tuples / (10 tuples/block) = 6000 blocks and
B(S) = 30000 tuples / (30 tuples/block) = 1000 blocks.
We can estimate that T(1.1) = T(2.1) = T(R) / V(R, A) = 60000 / 6 = 10000 and
T(2.2) = T(S) / V(S, C) = 30000 / 20 = 1500. This gives us
B(1.1) = B(2.1) = 10000 tuples / (10 tuples/block) = 1000 blocks and
B(2.2) = 1500 tuples / (30 tuples/block) = 50 blocks.
a)
We need 6000 I/O's to read R and execute the selection. We know from the task description that the resulting tuples are not written to disk but are directly handed over to the hash function to create the hash buckets.
We need 1000 I/O's to write these hash buckets to disk and again 1000 I/O's to read them from disk in the second phase of the hash join.
We need 1000 I/O's to read S and create the hash buckets.
We need another 1000 I/O's to write these hash buckets to disk and again 1000 I/O's to read them from disk in the second phase of the hash join.
In total we require 11000 I/O's.

b)
Similar to a) we need 8000 I/O's to process the data on the left branch of the query.
We need 1000 I/O's to read S and execute the selection. We know from the task description that the resulting tuples are not written to disk but are directly handed over to the hash function to create the hash buckets.
We need 50 I/O's to write these hash buckets to disk and again 50 I/O's to read them from disk in the second phase of the hash join.
In total we require 9100 I/O's.

2) Suppose B(R) = 20000, B(S) = 50000, and the number of main memory blocks is M = 101. We want to perform the natural join of R and S using a merge join algorithm. Both relations are clustered but none of them is sorted.
a) How many passes do we need?
b) Describe how the join is executed and how many I/O's are required.
Solution:
a) We have B(S) > B(R) and
$M^2 = 10201 \not> B(S)$. Therefore, 2 passes are not sufficient. But
$M^3 = 1030301 > B(S)$ and therefore, 3 passes will work and as also $M^3 > B(R) + B(S)$, we can use the optimized variant of the merge join.
b)
Pass 1:
For R we get $\lceil$(20000 blocks) / (101 blocks/sublist)$\rceil$ = 199 sublists
For S we get $\lceil$(50000 blocks) / (101 blocks/sublist)$\rceil$ = 496 sublists
Pass 2:
For R we get $\lceil$(199 sublists) / (100 sublists merged to 1)$\rceil$ = 2 sublists
For S we get $\lceil$(496 sublists) / (100 sublists merged to 1)$\rceil$ = 5 sublists
Pass 3: We read the 2 + 5 sublists and perform the actual join.
For pass 1 and 2 we have in each 2 I/O's per block and in pass 3 just 1 I/O per block. In total we need: 5 I/O's per block * (20000 + 50000) blocks = 350000 I/O's.

c:

M = min(B(R),B(S)) +1

we get: M = min(20000, 50000) +1 = 20000 + 1 = 20,001 blocks

A One Pass Join
If M>min(B(R),B(S)), the smaller relation can completely fit in the memory with one or more buffers to spare.

This means we can keep one relation entirely in memory, and compare and join with the other, by reading the other relation block by block into the spare buffers. This gives a one pass join.

In other words:
M-1 = min(B(R),B(S)) , ie,
M = min(B(R),B(S)) +1
is the minimum memory requirement for a trivial one pass join.
Cost of One Pass Join = B(R) + B(S)

3) We would like to sort the tuples of a relation R on a given key. The following information is known about the relation.

- The relation R contains 100000 tuples, i.e., T(R) = 100000.
- The size of a block on disk is 4000 bytes.
- The size of each R tuple is 400 bytes.
- Relation R is clustered, that is each disk block holding R tuples is full of R tuples.
- The size of the sort key is 32 bytes.
- A record pointer is 8 bytes.

Answer the following questions based on the information above.
a) If we use a two pass sorting algorithm, what is the minimum amount of main memory (in number of blocks) required?
b) What is the cost of the two pass sorting algorithm in terms of number of disk I/Os? Include the cost of writing the sorted file to disk at the end in your calculations.
c) Consider the following variant of the sorting algorithm. Instead of sorting the entire tuple, we just sort the <key, recordPointer> for each tuple. As in the conventional two pass sorting algorithm, we sort chunks of <key, recordPointer> in main memory and write the chunks to disk. In the merge phase, <key, recordPointer> entries from different chunks are merged. The record pointers are used to recover the rest of the tuple (from the original copy of R) and write the sorted relation to the disk. What is the cost in terms of number of disk I/Os?
Solution:
We have (4000 bytes/block) / (400 bytes/record) = 10 records/block and
B(R) = 100000 records / (10 records/block) = 10000 blocks
a) We know that with n main memory blocks we can sort a file with up to n(n-1) blocks. Therefore, it has to hold that n(n-1) ≥ B(R). The minimum value with this property is 101 blocks.
b) We need 4 I/O's per block so that we get 4 * B(R) = 40000.
c) The size of a key-pointer-pair is (32 + 8) bytes = 40 bytes.
We have (4000 bytes/block) / (40 bytes/key-pointer-pair) = 100 key-pointer-pairs/block and therefore, we need
100000 key-pointer-pairs / (100 key-pointer-pairs/block) = 1000 blocks to store all key-pointer-pairs. The variant of the algorithm then needs the following I/O's:
Read the relation and construct the key-pointer-pairs: 10000 I/O's
Write the sorted sublists of key-pointer-pairs to disk: 1000 I/O's
Read the sorted sublists of key-pointer-pairs from disk: 1000 I/O's
Retrieve the tuples pointed to by the record pointers: 100000 I/O's
Write the sorted relation to disk: 10000 I/O's
In total we get 122000 I/O's.