Implementation of DBMS
Exercise Sheet 5, Solutions
Klingemann, WS 2024 / 2025

1) Suppose that if we swizzle all pointers automatically, we can perform the swizzling in half the time it would take to swizzle each one separately. If the probability that a pointer in main memory will be followed at least once is `p`, for what values of `p` is it more efficient to swizzle automatically than on demand?

Solution:

Let `t` be the time to swizzle a pointer on demand. When we swizzle on demand, we have to invest this amount of time with a probability of `p`. Therefore, the average amount of time per pointer is `pt`. When we swizzle automatically, we have to invest for each pointer `t/2`. As a result, swizzling automatically is more efficient when `t/2 < pt`, i.e., `p > ½`.

2) Suppose blocks hold either ten records, or 50 key-pointer pairs. We have a data file with 10^6 records. In this task we assume that each block will be as full as possible.

a) How many blocks do we need for the data file?

b) How many blocks do we need for a dense index?

c) How many blocks do we need for a sparse index?

d) We want to add higher level index structures to the index considered in b). Describe how many blocks we have in these higher levels until we end up in a level with just one block.

e) Repeat d) for the sparse index in c).

Solutions:

a) $\lceil(10^6 \text{ records}) / (10 \text{ records / block})\rceil = 10^5$ blocks

b) $\lceil(10^6 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 20000$ blocks

c) $\lceil(10^5 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 2000$ blocks

d) 2nd level index: $\lceil(20000 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 400$ blocks
   3rd level index: $\lceil(400 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 8$ blocks
   4th level index: $\lceil(8 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 1$ block

e) 2nd level index: $\lceil(2000 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 40$ blocks
   3rd level index: $\lceil(40 \text{ key-pointer pairs}) / (50 \text{ key-pointer pairs / block})\rceil = 1$ block

3) Use the same scenario as in task 2. We assume that nothing is in memory initially, and that the search key is the primary key for the records. What is the average number of disk I/O's needed to retrieve a particular record that is stored in our data file given its search key using the following approaches?

a) We do not use any index but sequentially inspect the data file from the beginning until we find the record.

b) We just use the dense index described in task 2b to retrieve the record. We sequentially scan the index from the beginning until we find the search key value we are looking for.

c) We use the multi-level index described in task 2d to retrieve the record.

d) We use the multi-level index described in task 2e to retrieve the record.

Solutions:

a) It is equally likely that we find the record in any of the data blocks. Thus, the number of blocks to inspect is uniformly distributed. The best case is that we can stop after the first block. The worst case is that we need to inspect all 10^5 blocks. Thus, the average number of I/O’s is (1 I/O + 10^5 I/O’s) / 2 = 50000.5 I/O’s

b) Similar as in a), we sequentially scan blocks but this time index blocks. With the same reasoning as in a) have for the index blocks

(1 I/O + 20000 I/O’s) / 2 = 10000.5 I/O’s

The index block that we find contains a pointer to the record in the data file so that we need another I/O for the corresponding data block. As a result, we need on average 10001.5 I/O’s

c) We start with the block on the highest index level. This contains a pointer to a block in the next lower level that we have to follow. This continues until we reach the 1st level index so that we require 4 I/O’s for index blocks. Together with the I/O for the data block we need 5 I/O’s.

d) The approach is similar to c) be we have one index level less so that we need only 4 I/O’s.