CPSC 404: Advanced Relational Databases

Quiz 1 Key

Winter 2008, Term 2 Laks V.S. Lakshmanan

Q1(a) Since the file is not sorted on its primary key, any index on primary key will necessarily be unclustered.

(b) The number of data entries is exactly the number of records, since the primary key values of records will be distinct, by definition of primary key. Thus, the #data entries is 2 million.

(c) The size of 2 million data entries = 2 x 10^6 x (32 + 8) = 80 MB. At 8K/page, and assuming pages are filled to capacity, this corresponds to 80 x 10^6/8 x 10^3 = 10,000 pages.

(d) At 70% fill rate, the effective #leaves = ceil(10000/0.7) = 14,286. Max #children per internal node = 8K/(32+8) = 200. At 70% average fill rate, the average #children per internal node = 200 x 0.7 = 140. Thus, the height of the B+tree under this fill rate = ceil(log_140 14,286) = 2.

(e) See http://www.cs.ubc.ca/~laks/quiz1T2-2008KeyQ1e1.pdf and

http://www.cs.ubc.ca/~laks/quiz1T2-2008KeyQ1e2.pdf .

Q2(a) #SSLs produced in phase I = file size/buffer pool size = 22000/22 = 1000.

(b) While employing double-buffering, we need to allocate at least 2 pages per input and output buffer. Thus, we will be able to merge a maximum of 10 SSLs in one iteration. This requires (10 input buffers + 1 output buffer) x (2 pages/buffer) = 22 pages, which is exactly the size of the buffer pool.

(c) #iterations required = ceil(log_k #SSLs), where we do a k-way merge. For us, this works out to log_10 1000 = 3.

(d) In phase I, every page is read and written once. There are 3 passes in phase II. In each of them, every page is read and written once. So, in all, the total # page reads/writes = 2 x (1 + 3) x (22,000 pages) = 176,000.

(e) Consider any pass in phase II. Each buffer is allocated 2 pages. With double buffering, the max#pages (from any SSL) that get read sequentially **at a time** = 2/2 = 1. The output buffer is 2 pages as well, so by the same argument, the max #pages of any SSL that get written sequentially to disk **at a time** = 2/2 = 1. So, in every pass of phase II, any time a page is read or written, it's a random access. Thus, the total #random accesses incurred in phase II = 3 passes x 2 (for read + write) x 22,000 pages = 132,000.

Note: For the challenge question below, the solution provided here is more detailed than expected from you. It is so for pedagogical purposes.

CHALLENGE QUESTION: For Q2 above, under the current buffer allocation of 2 pages per buffer, cost of I/Os (just the transfer time) = 176,000 x 0.5 ms = 88 s. Cost of random accesses = 132,000 x 25 ms = 3300 s. The overall sorting cost = 3388 s.

The dominant term is the random access cost, which can be cut down by changing the #SSLs merged at a time (the "k" in k-way merge) and the #pages allocated per buffer. Here, k can only lie in the range [2, 10], since we need to merge at least 2 at a time and the maximum that can be merged at a time is 10, as seen in Q2(b). The #passes for various values of k are as follows:

k	#passes
2	10
3	7
4, 5	5
6, 7, 8, 9	4
10	3

For k=10 (3 passes), we already know the overall cost --- 3388 s.

For 4 passes, we could set k to one of 6, 7, 8, or 9. For all these values of k, the transfer cost is the same. The random access cost is #passes(22000/half input buffer + 22000/half output buffer) 25 ms = #passes(22/half input buffer + 22/half output buffer) 25 s. Thus, the key is minimizing the sum (22/half input buffer +

22/half output buffer). For k=9, the only option is 9 x 2 + 1 x 4 and leads to a sum of 22/1 + 22/2 = 33. For k=8, we can only have 8 x 2 + 1 x 6, giving a sum of 22/1 + ceil(22/3) = 30. For k=7, we can only do 7 x 2 + 1 x 8, which gives 22/1 + ceil(22/4) = 28. Finally, k=6 gives 6 x 2 + 1 x 10, and 22/1 + ceil(22/5) = 27, which is the best among values of k that enjoy 4 passes. It gives an overall cost of (1+4)2 x 22 x 0.5 + 4(27)25 s = 2810 s.

For 5 passes, we could use k=4 or k=5. With k=5, we can do one of 5 x 2 + 1 x 12 OR 5 x 4 + 1 x 2 and with k = 4, we can do one of 4 x 2 + 1 x 14 OR 4 x 4 + 1 x 6. The "sum" in each case is 22/1 + ceil(22/6) = 26, 22/2 + 22/1 = 33, 22/1 + ceil(22/7) = 26, and 22/2 + ceil(22/3) = 19. The last is the best, and yields and overall cost of (1 + 5)2 x 22 x 0.5 + 5(19)25 s = 2507 s.

There are no values of k for which we get 6, 8, or 9 passes (check!). For 7 passes, we can set k = 3. The possible buffer allocations are 3 x 2 + 1 x 16 (sum = 22/1 + ceil(22/8) = 25), 3 x 4 + 1 x 10 (sum = 22/2 + ceil(22/5) = 11 + 5 = 16), and 3 x 6 + 1 x 4 (sum = ceil(22/3) + 22/2 =19). The middle one is the best and gives an overall cost of (1 + 7) 22 x 2 x 0.5 + 7(16)25 s = 2976 s.

For 10 passes, we can set k = 2. Among the various buffer allocations for this k, the allocation 2 x 6 + 1 x 10 (sum = ceil(22/3) + ceil(22/5) = 13) is the best and has an overall cost of (1 + 10) 22 x 2 x 0.5 + 10 (13)25 s = 3492 s.

Clearly, the choice k = 4 (5 passes) with 4pages/input buffer 6 pages for the output buffer is the optimal choice. It has an overall cost of 2507 s.

> **FYI**: A formal proof of optimality (not required in this quiz) would involve expressing the overall sorting cost as a function of the #pages allocated per buffer and then using calculus for deriving the optimal value.