Database System Implementation

Solutions for Chapter 2

Solutions for Section 2.1

Exercise 2.1.1(a)

Terabyte disks are about 100 times as large as ``1999" disks. 100 is about 2^7, so there will have to be about seven doublings of disk size. According to Moore's law, that will take 10.5 years, so terabyte disks should be available in computer stores around 2010.

Return to Top

Solutions for Section 2.2

Exercise 2.2.1(a)

The disk has 10 * 10,000 = 100,000 tracks. The average track has 1000 * 512 = 512,000 bytes. Thus, the capacity is 51.2 gigabytes.

Exercise 2.2.1(c)

The maximum seek time occurs when the heads have to move across all the tracks. Thus, substitute 10,000 (really 9999) for n in the formula 1+.001n to get 11 milliseconds.

Exercise 2.2.1(d)

The maximum rotational latency is one full revolution. Since the disk rotates at 10,000 rpm, it takes 1/10000 of a minute, or 1/167 of a second to rotate, or about 6 milliseconds.

Exercise 2.2.3

First, we need to estabish the probability density function for the position of a random sector. Since these vary with the radius, the density at the inner edge is 3/7 the density of the outer edge, so the probability density looks like a trapezoid:

Call this function *p(x) = 3/5 + 4x/5*, with *x* varying from 0 to 1 and representing the fraction of the way from the inner to the outer edge that a sector is.

Our question asks for the average over all *x* and *y* distributed by function *p* of the magnitude of *x-y*, that is:

$$\int_0^1 \int_0^1 p(x)p(y)|x-y|\,dx\,dy$$

Since the moves outward must equal the moves inward, we can eliminate the absolute-magnitude operator by doubling, and restricting to the case where x>y. That is, we compute:

$$2\int_0^1 \int_0^x p(x)p(y)(x-y)\,dx\,dy$$

If we substitute for p, we get

$$2\int_0^1 \int_0^x (3/5 + 4x/5)(3/5+4y/5)(x-y)\,dx\,dy$$

Now, we have only to expand terms, integrate first on y, then on x, to get the value of this integral: 121/375. That is, the average sector-to-sector move of the heads will traverse this fraction of the tracks, which is just slightly less than 1/3 of the tracks.

[Return to Top]

Solutions for Section 2.3

Exercise 2.3.2(a)

The relation occupies 250,000 blocks, and the sort takes 4 disk I/O's per block, or 1,000,000 disk I/O's. If the number of tuples were doubled, the relation would occupy twice as many blocks, so we would expect 2,000,000 disk I/O's. We should check that the 2-phase sort is still possible, but since doubling the size of the relation would require 40 sublists rather than 20, there will still be plenty of room in memory on the second pass.

Exercise 2.3.2(c)

Doubling the size of blocks reduces the number of disk I/O's needed to half, or 500,000. We might imagine that we could keep growing the size of blocks indefinitely, and thus reduce the number of disk I/O's to almost zero. In a sense we can, but there are limitations, including the fact that transfer time will eventually grow to dominate other aspects of latency, in which case counting disk I/O's fails to be a good measure of running time.

Exercise 2.3.4

Binary search requires probing the block in the middle of the 250,000 blocks, then the one in the middle of the first or second half, and so on, for log_2(250,000) = 18 probes, until we can narrow down the presence of the desired record on one block. Thus, we require 18 disk I/O's.

Exercise 2.3.6

Look ahead to the solution to Exercise 2.3.8. The formula derived there tells us that:

1. *Doubling B halves n. That is, the larger the block size, the smaller the relation we can sort with the two-phase, multiway merge sort! Here is one good reason to keep block sizes modest.*

2. *If we double R, then we halve the number of tuples we can sort, but that is no surprise, since the number of blocks occupied by the relation would then remain constant.*

3. *If we double M we multiply by 4 the number of tuples we can sort.*

Exercise 2.3.8

The number of sorted sublists s we need is s = nR/M. On the second phase we need one block for each sublist, plus one for the merged list. Thus, we require Bs < M. Substituting for s, we get nRB/M < M, or n < M^2/RB.

[Return to Top]

Solutions for Section 2.4

Exercise 2.4.2

Recall that the Megatron 747 has a transfer time (in milliseconds) of 0.5, and an average rotational latency of 7.8. If the average seek moves 1/3 of the way across 4096 tracks, the average seek time for the Megatron 747 is 1 + .002(4096/3) = 3.7. Thus, the answer to part (a) is one block per 0.5 + 7.8 + 3.7 = 12 milliseconds, or 83 blocks per second, on each disk. Thus, the system can read 166 blocks per second.*

For a ``regular'' Megatron 747, the average latency is 0.5 + 7.8 + 6.5 = 14.8 milliseconds. However, if we have two, mirrored disks, each can be handling a request at the same time, or one read per 7.4 milliseconds. That gives us 135 blocks per second as the answer to (b).

For part (c), restricting each disk to half the tracks means that if there is not always a queue of waiting requests (which slows the applications that the disk is supporting), then there might be two requests for the same half of the disk, in which case one disk is idle and a request waits.

Exercise 2.4.3(a)

Suppose that there are n requests waiting on each pass in the steady state. Then the time taken for a pass of the elevator algorithm is:

1. *0.5n for the transfers.*

2. *7.8n for the rotational latencies.*

3. *n + 16.4 for the seek times. To see the latter, notice that the head must start n times, which accounts for the first term, and total travel is 8192 tracks, which accounts for the second term, at the Megatron 747's rate of 500 tracks per millisecond.*

Let the time taken for a pass be t. Then 0.5n + 7.8n + n +16.4 = t. Also, during time t, another n requests must arrive, and they arrive one every A milliseconds. Thus, t = An.

We can solve these two equations for n in terms of A; it is n = 16.4/(A - 9.3). The answer we want is t, the time to perform the pass. But t = An = 16.4A/(A - 9.3).

Exercise 2.4.4

Think of the requests as a random sequence of the integers 1 through 4. This sequence can be divided up into segments that do not contain two of the same integer, in a greedy way. For example, the sequence 123142342431 would be divided into 123, 1423, 42, and 31. The disks are serving all the requests from one segment, and each request is generated and starts at about the same time. When a segment is finished, the waiting request begins, along with other requests for other disks that are, by our assumption, generated almost immediately and thus finish at about the same time.

The question is thus: if I choose numbers 1, 2, 3, and 4 at random, how many choices, on the average, can I make without a repeat? The cases are:

1. *1/4 of the time we get an immediate repeat on the second choice, and the length is therefore 1.*

2. *(3/4)(1/2) = 3/8 of the time we get our first repeat at third number, and the length is 2.*

3. *(3/4)(1/2)(3/4) = 9/32 of the time we get our first repeat at the fourth number, and our length is 3.*

4. *The remaining fraction of the time, 3/32, we get four different numbers, and our length is 4.*

*The average length is therefore: (1/4)*1 + (3/8)*2 + (9/32)*3 + (3/32)*4 = 71/32.*

Return to Top

Solutions for Section 2.5

Exercise 2.5.1(a)

There are five 1's, which is odd, so the parity bit is 1.

Return to Top

Solutions for Section 2.6

Exercise 2.6.2

To compute the mean time to failure, we can compute the probability that the system fails in a given year. The MTTF will be the inverse of that probability. Note that there are 8760 hours in a year.

a) The system fails if the second disk fails while the first is being repaired. The probabiliy of a failure in a year is 2F. The probability that the second disk will fail during the H hours that the

other is being prepared is H/8760. Thus, the probability of a failure in any year is 2FH/8760, and the MTTF is 4380/FH

b) *The system loses data if there are three disk crashes within an H hour period. The probability of any one disk failing in a year is NF. If so, there are (N-1)(N-2)/2 pairs of other disks that could fail within an H hour period, and the chance that any one of them will do so is FH/8760. Thus, the probability of two additional disk failures is (N-1)(N-2)F^2H^2/(2*8760^2), or approximately (NFH)^2/1.5*10^8. The MTTF is the inverse of this probability.*

Exercise 2.6.4(a)

01010110.

Exercise 2.6.5(a)

01010110. Notice that this question calls for the same computation as Exercise 2.6.4(a), or at least it would have, except for the typo in the third ``block,'' where 0011011 should have been 00111011.

Exercise 2.6.8(a)

In the matrix of Fig. 2.17, we see that columns 1 and 7 differ in both the first and second rows. In each case, column 1 has bit 1 and column 7 has bit 0. Let's use the first row to recover disk 1. That step requires us to take the modulo-2 sum of disks 2, 3, and 5, the other disks where row 1 has bit 1.

Next, we use row 3, the only row where column 7 has a 1, to recover disk 7 by taking the modulo-2 sum of disks 1, 3, and 4.

Exercise 2.6.12

The parity checks of disks 9 and 10 can be represented by the matrix

1 1 1 1 0 0 0 0 1 0 0

0 0 0 0 1 1 1 1 0 1 0

Before we add a third row representing the parity check of disk 11, we can recover from two disk failures if and only if the two disks have different columns (except for disk 11, which has only 0's in its column). We'll surely add a 1 in column 11. So far, there are five columns with 1,0 (columns 1, 2, 3, 4, and 9) and five columns with 0,1 (columns 5, 6, 7, 8, and 10). The best we can do is break each of these groups 3-and-2, to maximize the number of pairs of disks that no longer have identical columns. Thus, we can make disk 11 be a parity check on any two or three of the first group and any two or three of the second group. An example would be to make disk 11 be a parity check on columns 3 through 6, but there are many other correct answers.

Return to Top