Database System Implementation

Solutions for Chapter 4

Solutions for Section 4.1

Solutions for Section 4.2

Solutions for Section 4.3

Solutions for Section 4.4

Solutions for Section 4.1

Revised 8/10/01.

Exercise 4.1.1

a) *n*/3 blocks are needed for the data file, and *n*/10 for the index, for a total of 13*n*/30 blocks.

b) We again need *n*/3 for the data file, but now need only room for *n*/3 pointers in the index file. The latter takes *n*/30 blocks, for a total of 11*n*/30 blocks.

Exercise 4.1.4

First, we need one disk I/O to find the place in the index that will tell us about key *K*, and then we need another disk I/O to read that index block. A third disk I/O takes us to the first data block with that key value.

Since there are at most three records with a given key value, at most we shall have to look at this block and the next in sequence. The question doesn't specify how that block is found, but let's assume that either the data blocks are consecutive on the disk, or that they are linked with pointers in a chain, so there is no additional cost to *finding* the next block, just an additional one disk I/O for reading it.

Now, we need to compute the probability that the records with key *K* extend over two blocks. 1/3 of the time, there is only one record, so the next block is surely not needed. Another 1/3 of the time, there are two records with that key, and 1/3 of those times, the first of the two records is at the end of its block. Only in that case will the second block be read. The last 1/3 of the time, there are three records with that key, and 2/3 of those will *not* have all three in the same block. Thus, the fraction of the time we shall do the 4th disk I/O is (1/3)*(1/3) + (1/3)*(2/3) = 1/3. The answer is thus 10/3 disk I/O's.

Exercise 4.1.7(a)

The first index block has keys 10, 20, and 50, plus an empty slot. The data record with 50 and 60 just has 50. The data record with 70 and 80 is deleted. The data record with 20 and 40 is replaced by a chain of blocks with the following contents: (20,21), (22,23), (24,25), (26,27), (28,29), and (40,-).

Exercise 4.1.8

First, notice that we search an average of 2 blocks on a chain when the chain has 3 full blocks. Thus, the question really asks when the average number of records on each chain is enough to fill 3 blocks. Since initially, each ``chain'' has one half-full block, the number of records must grow by a factor of 6; i.e., when there are 6*n* records.

Return to Top

Solutions for Section 4.2

Exercise 4.2.1

When we insert a record with key *K*, we must insert a record into the secondary index file with that value *K* and a pointer to the data record. Any technique for inserting into a sequential file is appropriate.

When we delete a record with key *K*, we need to locate the corresponding record in the secondary-index file. Since there may be duplicate keys, any method for locating the record(s) with key *K* in a sequential file with duplicate keys will work. When we find these records in the secondary-index file, we need to find the one that points to the record being deleted; i.e., we must examine each of them until we find the one with a pointer to the correct record. We then delete this record from the secondary-index file as we would from any sequential file.

Exercise 4.2.3

This question asks what is the average number of blocks over which $m+1$ records extend, if 10 fit in a block, and they start at a random slot on a block. If $m = 0$, the answer is clearly 1, and if $m = 10$, the answer is clearly 2. As the function must be linear in m, we can infer that the average is always $m/10$.

If the records were distributed on different blocks, the number of disk I/O's would be $m+1$.

Exercise 4.2.4(a)

We need 1000 data blocks. If we use buckets, we need key-pointer pairs for only the 300 different keys. The latter fits on 30 index blocks. In addition, we need 3000 pointers in the buckets, which fits in 60 blocks, for a total of 1090 blocks.

If we do not use buckets, we need the same 1000 data blocks, but now we need a key-pointer pair for each of the 3000 records. These require 300 blocks, for a total of 1300.

Exercise 4.2.6(a)

To get the Disney movies, we need one disk I/O to get the secondary index block for key Disney. No matter where the 51 pointers begin, they will cover 2 bucket blocks, so another two disk I/O's are needed. Similarly, to get the 101 pointers to movies of 1995 requires a total of four disk I/O's, since the pointers must cover exactly three bucket blocks.

We intersect these pointers in memory, and since there is only one pointer in the intersection, an 8th disk I/O gets the desired record.

Exercise 4.2.7(a)

The total number of word occurrences is the sum from $i = 1$ to 10,000 of the number of occurrences of word i, which is 100,000/*sqrt(i)*. That number can be approximated by the integral *INT 100,000/sqrt(i)* from i=0 to 10,000. The latter is 200,000*sqrt*(10,000), or 20,000,000. Since there are 1000 documents, the average document has 20,000 words.

Exercise 4.2.7(b)

Since even the least frequent word appears 100,000/*sqrt*(10,000) = 1000 times, the factor that limits the size of the index for each word is that there are only 1000 documents. That is, we expect most of the 10,000 words to appear in most documents, which makes sense because these are really long documents, about 1/10 the size of the average book.

Thus, we need at most 20 blocks per word to hold up to 1000 pointers to documents, and we need for each word, one word/pointer pair that gets us to the first block of pointers for that word. The latter are stored 10 to a block, so we need 1000 blocks for 10,000 words. In addition, we need 20 blocks for each of the 10,000 words for the pointers to its documents, for a grand total of 201,000 blocks.

Exercise 4.2.8(a)

Obtain all the bucket entries for ``cat" and ``dog." Sort these by type, and within type, by position. Scan the records, keeping a ``window" with records of the current type, extending up to five positions prior to the current type. Compare each new record with the records in the window. If we find one that

a) Has the opposite word, e.g., ``dog" if the current bucket record has ``cat," and

b) have identical document entries,

then the common document is one of those we want to retrieve.

Return to Top

Solutions for Section 4.3

Exercise 4.3.1(a)

Note that part (a) contradicts the stem of the exercise by claiming the sequential file has 100 records/block. We shall assume 10 records/block, as in the stem, is correct.

First, there are 100,000 data blocks. If there are an average of 70 pointers per block in the bottom-level nodes of the B-tree, then there are 14286 B-tree blocks at that level. The next level of the B-tree requires 1/70th of that, or 2041 blocks, and the third level has 1/70th of that, or 30 blocks. The fourth level has only the root block. The number of blocks needed is therefore 100,000 + 14,286 + 2041 + 30 + 1 = 116,358 blocks.

Since the B-tree has four levels, we must make a total of five disk reads to go through the B-tree to the desired data block.

Exercise 4.3.1(e)

To begin, there are 1,000,000/15 primary blocks, or 66,667 primary blocks, each with 10 records, and the same number of overflow blocks, each with 5 records.

The number of first-level B-tree blocks is 66,667/70 = 953. At the second level there are 953/70 = 14, and the third level has only the root. Thus, there are 133,334 data blocks and 968 index blocks, for a total of 134,302 blocks.

A search requires three disk I/O's to go through the B-tree to the data file. Once there, we surely need to read the primary block, and 1/3 of the time we shall need to see the overflow block as well. Thus, the average number of disk I/O's is 13/3.

Exercise 4.3.4(a)

Interior nodes: at least 5 keys and 6 pointers. Leaves: at least 5 keys and pointers, not counting the optional extra pointer to the next leaf block.

Exercise 4.3.7(a)

Follow the standard lookup procedure to find (a pointer to) the first record with the given key. Then, continue through this leaf block of the B-tree, and any other leaves to its right, if necessary, searching for occurrences of the same key. We may stop as soon as a different key is seen. To find leaf blocks to the right, follow the pointers at the end of each leaf block.

Exercise 4.3.9(a)

First, the six data records can be divided among the B-tree leaves in the following ways: 2-2-2 or 3-3. There are no other ways to divide size records, with 2 or 3 for each leaf.

In each of these two cases, there can be only one other node, the root, and it must point to each of the two or three leaves. Again, there are no other ways to structure 2 or 3 leaves, with each interior node having 2 or 3 children. Our conclusion is that there are only two different B-trees for 6 data records.

Exercise 4.3.10

Since we are always adding at the right extreme of the B-tree, once split, the first (left) of the pair never changes. Thus, the leaf blocks will each have two keys only: (1,2), (3,4), (5,6), and so on.

At the second level, we divide pointers 3 to the left, and 2 to the right, so each except the rightmost node at that level has three pointers; the leftmost has two. The same holds for any other level.

Thus, when we first get four levels, the root has two pointers. At the third level, there are blocks with 3 and 2 pointers, respectively. At the second level, there are five blocks, with 3, 3, 3, 3, and 2 pointers. These 14 pointers go to leaf blocks with two pointers each, for a total of 28 pointers to records.

Thus, when the record with key 28 is inserted, the 4th level is created.

Return to Top

Solutions for Section 4.4

Exercise 4.4.3(a)

Hashing the key still gets us to the bucket where all records with that key live. Insertion can be done without regard to duplicate keys. However, for lookup or deletion, once we find one record with that key, we have to continue looking for others in the same bucket.

The principal problem that occurs is when there are so few different keys that there are more buckets than keys. In that case, portions of the hash table will be unused, and buckets will grow in size, regardless of how many buckets we choose for our hash table.

Exercise 4.4.4(a)

Suppose $B = 10$. Since any integer can be written in the form $10a+b$, where $0 <= b < 10$, the bucket of any such integer can be determined by considering b only (since its square is $100a^2 + 20ab + b^2$, which modulo 10 is the same as b^2 modulo 10.

However, the squares of 0 through 9, modulo 10, are: 0, 1, 4, 9, 6, 5, 6, 9, 4, and 1. Thus, buckets 2, 3, 7, and 8 never get any records, while 1, 4, 6, and 9 get a double helping.

Exercise 4.4.6(a)

When we insert 0011, there are four records for bucket 0, which overflows. Adding a second bit to the bucket addresses doesn't help, because the first four records all begin with 00. Thus, we go to $i = 3$ and use the first three bits for each bucket address. Now, the records divide nicely, and by the time 1111 is inserted, we have two records in each bucket.

Exercise 4.4.7

1. We could leave forwarding addresses in buckets when we split them.

2. We could use the full sequence of bits produced by the hash function as the pointer, and use the extensible or linear hash table to look up a record whenever an external pointer was followed.

Exercise 4.4.9

In the best case, all buckets will have a number of records that is divisible by 100. Then, the number of blocks needed is 10,000.

However, in the worst case, we could have only one record in each of 999 buckets. Since we are not sharing blocks among buckets, we would need 999 blocks for these buckets. The remaining bucket has 999,001 records, which requires 9991 blocks, for a total of 10,990 blocks.

Return to Top