CHAPTER 13: DISK STORAGE, BASIC FILE STRUCTURES, AND HASHING

Answers to Selected Exercises

13.23 Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size B=512 bytes, interblock gap size G=128 bytes, number of blocks per track=20, number of tracks per surface=400. A disk pack consists of 15 double-sided disks.

(a) What is the total capacity of a track and what is its useful capacity (excluding interblock gaps)?

(b) How many cylinders are there?

(c) What is the total capacity and the useful capacity of a cylinder?

(d) What is the total capacity and the useful capacity of a disk pack?

(e) Suppose the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what is the transfer rate in bytes/msec and the block transfer time btt in msec? What is the average rotational delay rd in msec? What is the bulk transfer rate (see Appendix B)?

(f) Suppose the average seek time is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block given its block address?

(g) Calculate the average time it would take to transfer 20 random blocks and compare it with the time it would take to transfer 20 consecutive blocks using double buffering to save seek time and rotational delay.

Answer:

(a) Total track size = 20 * (512+128) = 12800 bytes = 12.8 Kbytes
Useful capacity of a track = 20 * 512 = 10240 bytes = 10.24 Kbytes

(b) Number of cylinders = number of tracks = 400

(c) Total cylinder capacity = 15*2*20*(512+128) = 384000 bytes = 384 Kbytes
Useful cylinder capacity = 15 * 2 * 20 * 512 = 307200 bytes = 307.2 Kbytes

(d) Total capacity of a disk pack = 15 * 2 * 400 * 20 * (512+128)
= 153600000 bytes = 153.6 Mbytes
Useful capacity of a disk pack = 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

(e) Transfer rate tr= (total track size in bytes)/(time for one disk revolution in msec)
tr= (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512 bytes/msec
block transfer time btt = B / tr = 512 / 512 = 1 msec
average rotational delay rd = (time for one disk revolution in msec) / 2 = 25 / 2
= 12.5 msec
bulk transfer rate btr= tr * (B/(B+G)) = 512*(512/640) = 409.6 bytes/msec

(f) average time to locate and transfer a block = s+rd+btt = 30+12.5+1 = 43.5 msec

(g) time to transfer 20 random blocks = 20 * (s + rd + btt) = 20 * 43.5 = 870 msec
time to transfer 20 consecutive blocks using double buffering = s + rd + 20*btt
= 30 + 12.5 + (20*1) = 62.5 msec
(a more accurate estimate of the latter can be calculated using the bulk transfer rate as follows: time to transfer 20 consecutive blocks using double buffering
= s+rd+((20*B)/btr) = 30+12.5+ (10240/409.6) = 42.5+ 25 = 67.5 msec)

13.24 A file has r=20000 STUDENT records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), MAJORDEPTCODE (4 bytes), MINORDEPTCODE (4 bytes), CLASSCODE (4 bytes, integer), and DEGREEPROGRAM (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk whose parameters are given in Exercise 4.18.

(a) Calculate the record size R in bytes.

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.

(c) Calculate the average time it takes to find a record by doing a linear search on the file if (i) the file blocks are stored contiguously and double buffering is used, and (ii) the file blocks are not stored contiguously.

(d) Assume the file is ordered by SSN; calculate the time it takes to search for a record given its SSN value by doing a binary search.

Answer:

(a) R = (30 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 4 + 3) + 1 = 113 bytes

(b) bfr = floor(B / R) = floor(512 / 113) = 4 records per block
b = ceiling(r / bfr) = ceiling(20000 / 4) = 5000 blocks

(c) For linear search we search on average half the file blocks= 5000/2= 2500 blocks.
i. If the blocks are stored consecutively, and double buffering is used, the time to read 2500 consecutive blocks
= s+rd+(2500*(B/btr))= 30+12.5+(2500*(512/409.6))
= 3167.5 msec = 3.1675 sec
(a less accurate estimate is = s+rd+(2500*btt)= 30+12.5+2500*1= 2542.5 msec)
ii. If the blocks are scattered over the disk, a seek is needed for each block, so the time is: 2500 * (s + rd + btt) = 2500 * (30 + 12.5 + 1) = 108750 msec = 108.75 sec

(d) For binary search, the time to search for a record is estimated as:
ceiling(log 2 b) * (s +rd + btt)
= ceiling(log 2 5000) * (30 + 12.5 + 1) = 13 * 43.5 = 565.5 msec = 0.5655 sec

13.25 Suppose only 80% of the STUDENT records from Exercise 13.24 have a value for PHONE, 85% for
MAJORDEPTCODE, 15% for MINORDEPTCODE, and 90% for DEGREEPROGRAM, and we use a variable-length record file. Each record has a 1-byte field type for each field occurring in the record, plus the 1-byte deletion marker and a 1-byte end-ofrecord marker. Suppose we use a spanned record organization, where each block has a 5-byte pointer to the next block (this space is not used for record storage).

(a) Calculate the average record length R in bytes.

(b) Calculate the number of blocks needed for the file.

Answer:

(a) Assuming that every field has a 1-byte field type, and that the fields not mentioned above (NAME, SSN, ADDRESS, BIRTHDATE, SEX, CLASSCODE) have values in every record, we need the following number of bytes for these fields in each record, plus 1 byte for the deletion marker, and 1 byte for the end-of-record marker:
R fixed = (30+1) + (9+1) + (40+1) + (8+1) + (1+1) + (4+1) +1+1 = 100 bytes
For the fields (PHONE, MAJORDEPTCODE, MINORDEPTCODE DEGREEPROGRAM), the average number of bytes per record is:
R variable = ((9+1)*0.8)+((4+1)*0.85)+((4+1)*0.15)+((3+1)*0.9)
= 8+4.25+0.75+3.6= 16.6 bytes
The average record size R = R fixed + R variable = 100 + 16.6 = 116.6 bytes
The total bytes needed for the whole file = r * R = 20000 * 116.6 = 2332000 bytes

(b) Using a spanned record organization with a 5-byte pointer at the end of each block, the bytes available in each block are (B-5) = (512 - 5) = 507 bytes.
The number of blocks needed for the file are:
b = ceiling((r * R) / (B - 5)) = ceiling(2332000 / 507) = 4600 blocks
(compare this with the 5000 blocks needed for fixed-length, unspanned records in Problem 4.19(b))

13.26 Suppose that a disk unit has the following parameters: seek time s=20 msec; rotational delay rd=10 msec; block transfer time btt=1 msec; block size B=2400 bytes; interblock gap size G=600 bytes. An EMPLOYEE file has the following fields: SSN, 9 bytes; LASTNAME, 20 bytes; FIRSTNAME, 20 bytes; MIDDLE INIT, 1 byte; BIRTHDATE, 10 bytes; ADDRESS, 35 bytes); PHONE, 12 bytes); SUPERVISORSSN, 9 bytes; DEPARTMENT, 4 bytes; JOBCODE, 4 bytes; deletion marker, 1 byte. The EMPLOYEE file has r=30000 STUDENT records, fixed-length format, and unspanned blocking. Write down appropriate formulas and calculate the following values for the above EMPLOYEE file:

(a) The record size R (including the deletion marker), the blocking factor bfr, and the number of disk blocks b.

(b) Calculate the wasted space in each disk block because of the unspanned organization.

(c) Calculate the transfer rate tr and the bulk transfer rate btr for this disk (see Appendix B for definitions of tr and btr).

(d) Calculate the average number of block accesses needed to search for an arbitrary record in the file, using linear search.

(e) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are stored on consecutive disk blocks and double buffering is used.

(f) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are not stored on consecutive disk blocks.

(g) Assume that the records are ordered via some key field. Calculate the average number of block accesses and the average time needed to search for an arbitrary record in the file, using binary search.

Answer:

(a) R = (9 + 20 + 20 + 1 + 10 + 35 + 12 + 9 + 4 + 4) + 1 = 125 bytes
bfr = floor(B / R) = floor(2400 / 125) = 19 records per block
b = ceiling(r / bfr) = ceiling(30000 / 19) = 1579 blocks

(b) Wasted space per block = B - (R * Bfr) = 2400 - (125 * 19) = 25 bytes

(c) Transfer rate tr= B/btt = 2400 / 1 = 2400 bytes/msec
bulk transfer rate btr= tr * (B/(B+G))
= 2400*(2400/(2400+600)) = 1920 bytes/msec

(d) For linear search we have the following cases:
i. search on key field:
if record is found, half the file blocks are searched on average: b/2= 1579/2 blocks
if record is not found, all file blocks are searched: b = 1579 blocks
ii. search on non-key field:
all file blocks must be searched: b = 1579 blocks

(e) If the blocks are stored consecutively, and double buffering is used, the time to read n consecutive blocks= s+rd+(n*(B/btr))
i. if n=b/2: time = 20+10+((1579/2)*(2400/1920))= 1016.9 msec = 1.017 sec
(a less accurate estimate is = s+rd+(n*btt)= 20+10+(1579/2)*1= 819.5 msec)
ii. if n=b: time = 20+10+(1579*(2400/1920))= 2003.75 msec = 2.004 sec
(a less accurate estimate is = s+rd+(n*btt)= 20+10+1579*1= 1609 msec)

(f) If the blocks are scattered over the disk, a seek is needed for each block, so the time to search n blocks is: n * (s + rd + btt)
i. if n=b/2: time = (1579/2)*(20+10+1)= 24474.5 msec = 24.475 sec
ii. if n=b: time = 1579*(20+10+1)= 48949 msec = 48.949 sec

(g) For binary search, the time to search for a record is estimated as:
ceiling(log 2 b) * (s +rd + btt)
= ceiling(log 2 1579) * (20+10+1) = 11 * 31 = 341 msec = 0.341 sec

13.27 A PARTS file with Part# as hash key includes records with the following Part# values: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981, 9208. The file uses 8 buckets, numbered 0 to 7. Each bucket is one disk block and holds two records. Load these records into the file in the given order using the hash function h(K)=K mod 8. Calculate the average number of block accesses for a random retrieval on Part#.

Answer:

The records will hash to the following buckets:
K h(K) (bucket number)

2369 1
3760 0
4692 4
4871 7
5659 3
1821 5
1074 2
7115 3
1620 4
2428 4 overflow
3943 7
4750 6
6975 7 overflow
4981 5
9208 0
9209

Two records out of 15 are in overflow, which will require an additional block access. The other records require only one block access. Hence, the average time to retrieve a random record is:
(1 * (13/15)) + (2 * (2/15)) = 0.867 + 0.266 = 1.133 block accesses

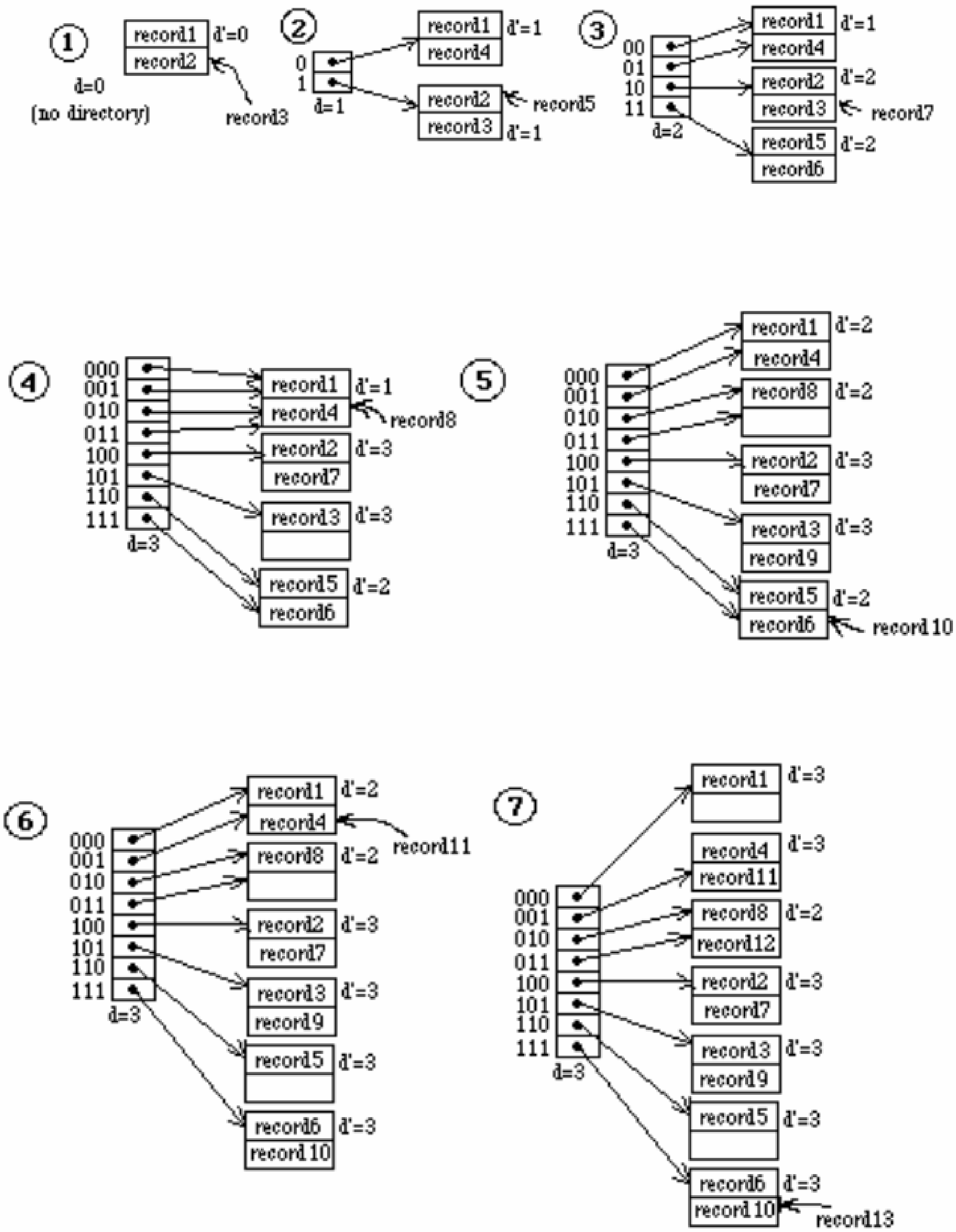
13.28 Load the records of Exercise 5.27 into expandable hash files based on extendible hashing. Show the structure of the directory at each step. Show the directory at each step, and the global and local depths. Use the has function
h(k) = K mod 32.

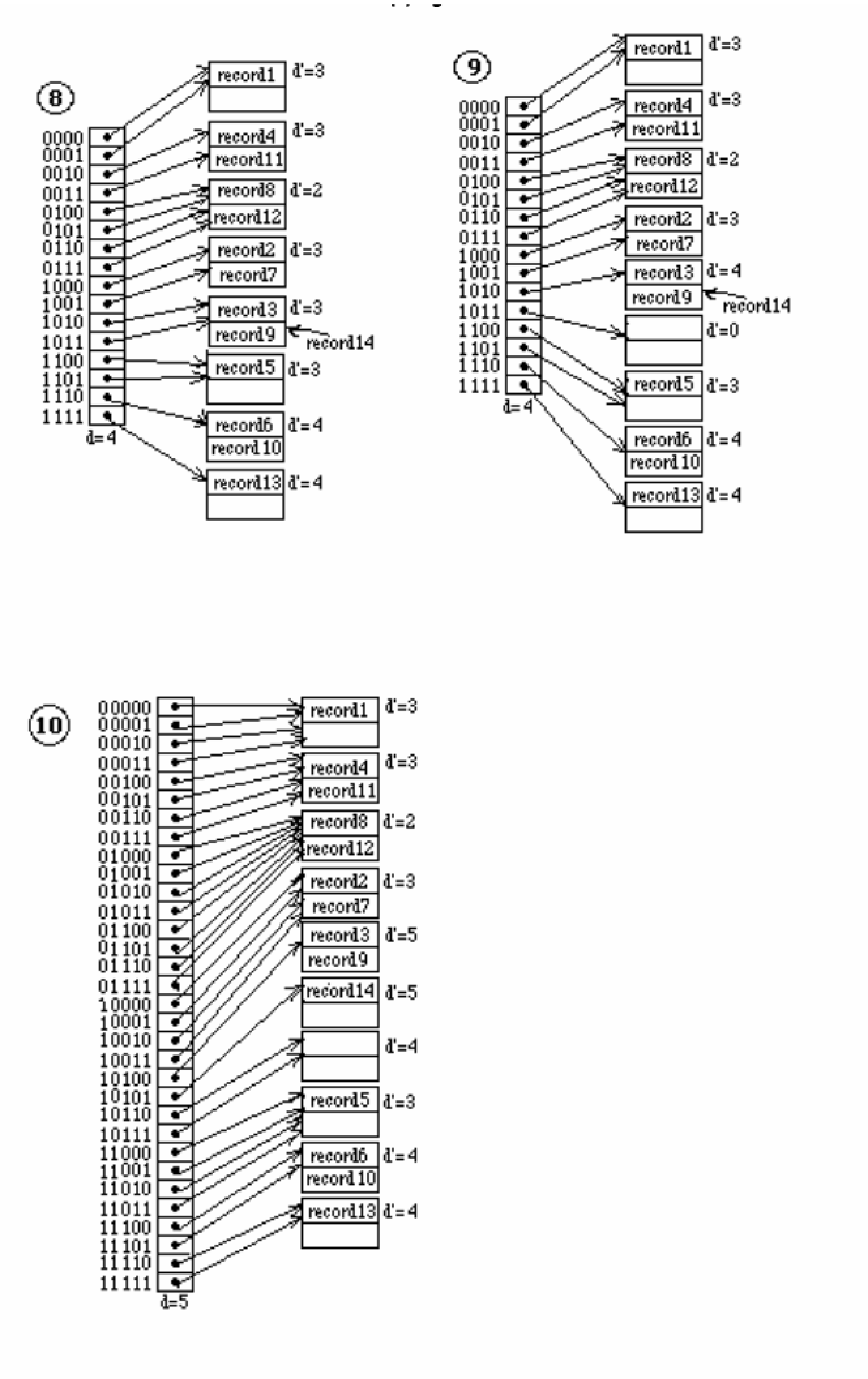
Answer:

Hashing the records gives the following result:

	K	h(K) (bucket number)	binary h(K)
record1	2369	1	00001
record2	3760	16	10000
record3	4692	20	10100
record4	4871	7	00111
record5	5659	27	11011
record6	1821	29	11101
record7	1074	18	10010
record8	7115	11	01011
record9	1620	20	10100
record10	2428	28	11100
record11	3943	7	00111
record12	4750	14	01110
record13	6975	31	11111
record14	4981	21	10101
record15	9208	24	11000

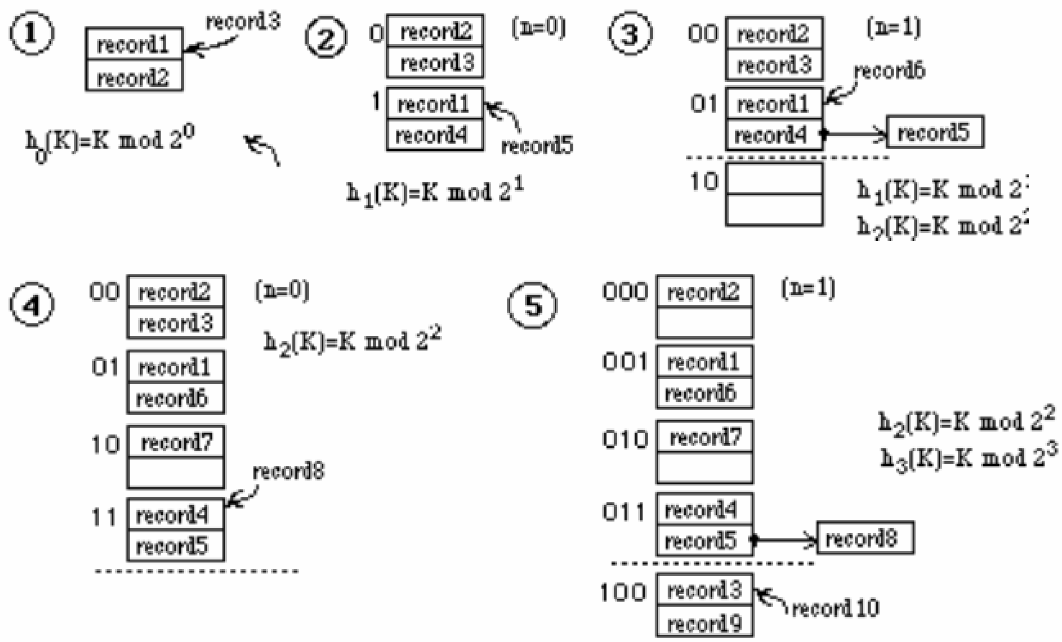
Extendible hashing:

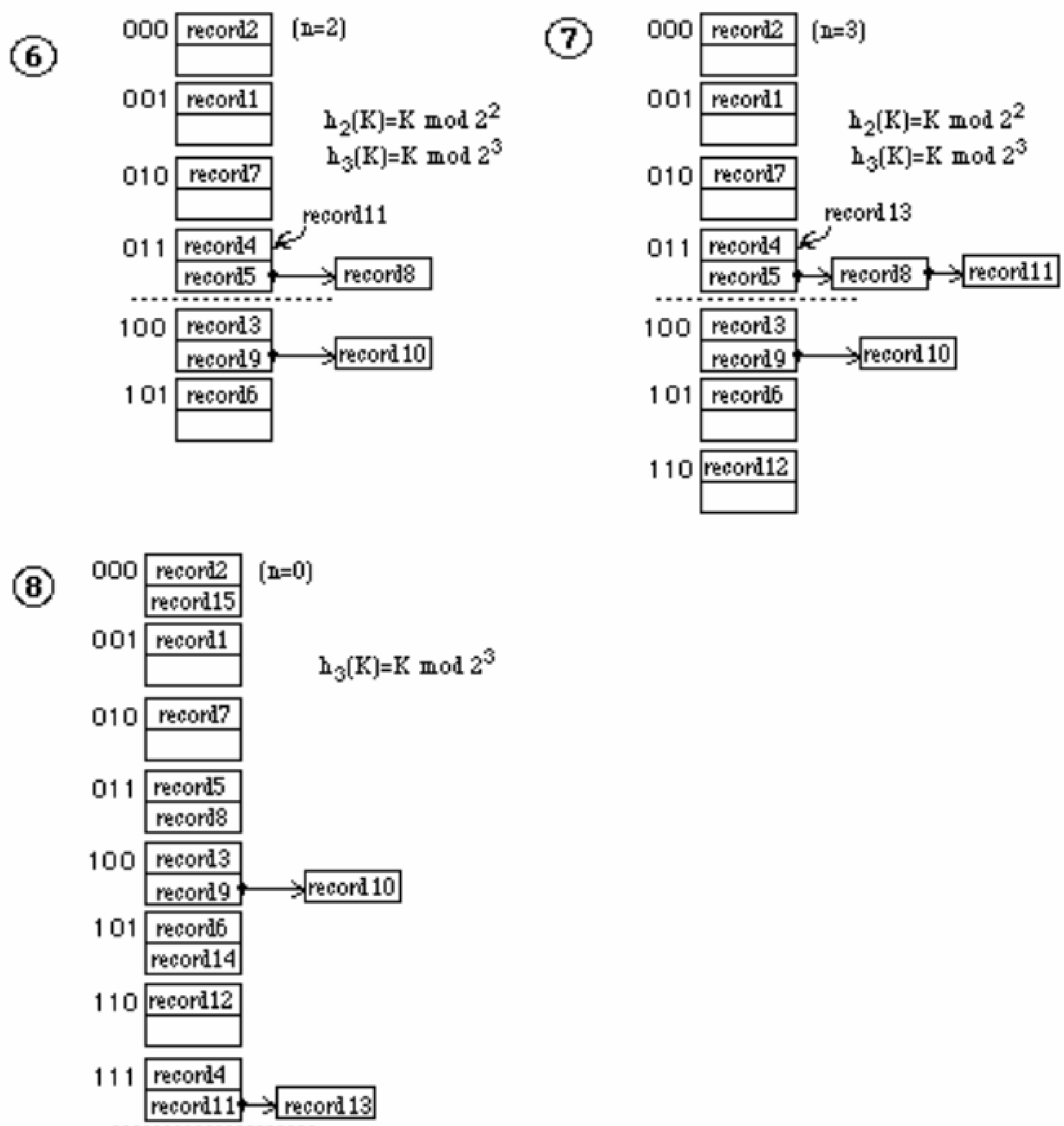



8
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d= 4
record1 d'=3
record4 record11 d'=3
record8 record12 d'=2
record2 record7 d'=3
record3 record9 d'=3
record14
record5 d'=3
record6 record10 d'= 4
record13 d'= 4
9
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d= 4
record1 d'=3
record4 record11 d'=3
record8 record12 d'=2
record2 record7 d'=3
record3 record9 d'= 4
record14
d'=0
record5 d'=3
record6 record10 d'= 4
record13 d'= 4
10
00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11101
11110
11111
d=5
record1 d'=3
record4 record11 d'=3
record8 record12 d'=2
record2 record7 d'=3
record3 record9 d'=5
record14 d'=5
d'=4
record5 d'=3
record6 record10 d'= 4
record13 d'= 4

13.29 Load the records of Exercise 13.27 into expandable hash files based on linear hashing.
Start with a single disk block, using the hash function *h* ø = *K* mod 2º, and show how the file grows and how the hash functions change as the records are inserted. Assume that blocks are split whenever an overflow occurs, and show the value of n at each stage

Answer:





Note: It is more common to specify a certain load factor for the file for triggering the splitting of buckets (rather than triggering the splitting whenever a new record being inserted is placed in overflow). The load factor lf could be defined as:
lf = (r) / (b * Bfr)

where r is the current number of records, b is the current number of buckets, and Bfr is the maximum number of records per bucket. Whenever lf gets to be larger than some threshold, say 0.8, a split is triggerred. It is also possible to merge backets in the reverse order in which they were created; a merge operation would be triggerred whenever lf becomes less than another threshold, say 0.6.

13.30 No solution provided

13.31 No solution provided

13.32 No solution provided

13.33 Can you think of techniques other than an unordered overflow file that can be used to make insertion in an ordered file more efficient?

Answer:

It is possible to use an overflow file in which the records are chained together in a manner similar to the overflow for static hash files. The overflow records that should be inserted in each block of the ordered file are linked together in the overflow file, and a pointer to the first record in the chain (linked list) is kept in the block of the main file. The list may or may not be kept ordered.

13.34 No solution provided

13.35 Can you think of techniques other than chaining to handle bucket overflow in external hashing?

Answer:

One can use techniques for handling collisions similar to those used for internal hashing. For example, if a bucket is full, the record which should be inserted in that bucket may be placed in the next bucket if there is space (open addressing). Another scheme is to use a whole overflow block for each bucket that is full. However, chaining seems to be the most appropriate technique for static external hashing.

13.36 No solution provided.

13.37 No solution provided.

13.38 Suppose that a file initially contains r=120000 records of R=200 bytes each in an unsorted (heap) file. The block size B=2400 bytes, the average seek time s=16 ms, the average rotational latency rd=8.3 ms and the block transfer time btt=0.8 ms. Assume that 1 record is deleted for every 2 records added until the total number of active records is 240000.

(a) How many block transfers are needed to reorganize the file?

(b) How long does it take to find a record right before reorganization?
(c) How long does it take to find a record right after reorganization?

Let X = # of records deleted
Hence 2X= # of records added.

Total active records
= 240,000 = 120,000 - X + 2X.
Hence, X = 120,000
Records before reorganization (i.e., before deleting any records physically) =
360,000.

(a) No. of blocks for Reorganization
= Blocks Read + Blocks Written.
-200 bytes/record and 2400 bytes/block gives us 12 records per block
-Reading involves 360,000 records; i.e. 360,000/12 = 30K blocks
-Writing involves 240,000 records; i.e., 240000/12 = 20K blocks.
Total blocks transferred during reorganization
= 30K + 20K = 50K blocks.

(b) Time to locate a record before reorganization. On an average we assume that
half the file will be read.
Hence, Time = (b/2)* btt = 15000 * 0.8 ms = 12000 ms.
= 12 sec.

(c) Time to locate a record after reorganization
= (b/2) * btt = 10000 * 0.8 = 8 sec.

13.39 Suppose we have a sequential (ordered) file of 100000 records where each record is 240 bytes. Assume that B=2400 bytes, s=16 ms, rd=8.3 ms, and btt=0.8 ms. Suppose we want to make X independent random records from the file. We could make X random block reads or we could perform one exhaustive read of the entire file looking for those X records. The question is to decide when it would be more efficient to perform one exhaustive read of the entire file than to perform X individual random reads. That is, what is the value for X when an exhaustive read of the file is more efficient than random X reads? Develop this function of X.

Total blocks in file = 100000 records * 240 bytes/record divided by 2400
bytes/block = 10000 blocks.
Time for exhaustive read
= s + r + b.btt
= 16 + 8.3 + (10000) * 0.8
= 8024.3 msec
Let X be the # of records searched randomly that takes more time than
exhaustive read time.
Hence, X (s + r + btt) > 8024.3
X (16+8.3+0.8) > 8024.3
X > 8024.3/25.1
Thus, X > 319.69
i.e. If at least 320 random reads are to be made, it is better to search the file
exhaustively.

13.40 Suppose that a static hash file initially has 600 buckets in the primary area and that records are inserted that create an overflow area of 600 buckets. If we reorganize the hash file, we can assume that the overflow is eliminated. If the cost of reorganizing the file is the cost of the bucket transfers (reading and writing al of the buckets) and the only periodic file operation is the fetch operation, then how many times would we have to perform a fetch (successfully) to make the reorganization cost-effective? That is, the reorganization cost and subsequent search cost are less than the search cost before reorganization. Assume s=16, rd=8.3 ms, btt=1 ms.

Primary Area = 600 buckets
Secondary Area (Overflow) = 600 buckets
Total reorganization cost = Buckets Read & Buckets Written for (600 & 600) + 1200 = 2400 buckets = 2400 (1 ms) = 2400 ms
Let X = number of random fetches from the file.
Average Search time per fetch = time to access (1 + 1/2) buckets where 50% of time we need to access the overflow bucket.
Access time for one bucket access = (S + r + btt)
= 16 + 8.3 + 0-8
= 25.1 ms
Time with reorganization for the X fetches
= 2400 + X (25.1) ms
Time without reorganization for X fetches = X (25.1) (1 + 1/2) ms
= 1.5 * X * (25.1) ms.
Hence, 2400 + X (25.1) < (25.1) * (1.5X)
2374.9/ 12.55 < X
Hence, 189.23 < X
If we make at least 190 fetches, the reorganization is worthwhile.

13.41 Suppose we want to create a linear hash file with a file load factor of 0.7 and a blocking factor of 20 records per bucket, which is to contain 112000 records initially.

(a) How many buckets should we allocate in primary areas?

(b) What should be the number of bits used for bucket addresses?

Answer:

(a) No. of buckets in primary area.
= 112000/(20*0.7) = 8000.

(b) K: the number of bits used for bucket addresses
2K < = 8000 < = 2 k+1
2 12 = 4096
2 13 = 8192
K = 12
Boundary Value = 8000 - 2 12
= 8000 - 4096
= 3904 -