Tutorial CS3402
File Organization & Indexing

1. Consider the extendible hashing scheme. Recall that we consider the binary representation of the hash values for assigning items to buckets. Suppose that the *global depth* is d=2 and that each bucket can hold 2 items. Initially, there are two (empty) buckets b1 and b2 that have a *local depth* d'=1, where
   - b1 is the bucket for records whose binary hash values start with 0, and
   - b2 is the bucket for records whose binary hash values start with 1.

   Below is the initial state of the directory:

Directory entry (d=2)	Pointer to Bucket
00	b1
01	b1
10	b2
11	b2

a) What is the content of buckets b1 and b2 after inserting items with hash values 0100, 0000, 1000, 1100? Are there any changes to the pointers in the directory?

b1 will hold 0100 and 0000. b2 will hold 1000 and 1100. No changes to directory.

b) Describe the steps that take place if we next insert an item with hash value 1110.

Solution: Bucket b2 overflows. To handle the overflow, we need to split the bucket into buckets b2 and b2' that both have a local depth of 2. b2 will hold items starting with 10 whereas b2' will hold items starting with 11. The directory now has 2 pointers to b1 and one pointer each to b2 and b2'.

c) After performing the insertions mentioned in a) and b), we next insert items with hash values 1101 and 1111. Describe how these insertions are handled. What is the final state of the directory and the contents of the buckets?

Solution: Bucket b2' will overflow. Since the local depth of b2' is 2 which is also the global depth, we must increase the global depth to 3. This means that the directory now has 2^3 = 8 entries. In addition, we split b2' into b2' and b2'', where b2' holds items starting with 110 and b2'' holds items starting with 111. The directory looks as follows:

Directory entry (d=3)	Pointer to Bucket
000	b1
001	b1
010	b1
011	b1
100	b2
101	b2
110	b2'
111	b2''

2. Suppose that you have formatted your disk with a block size of 1024 bytes and assume that we have 40,000 STAFF records of fixed length. A block pointer is P=6 bytes long, and a record pointer is Pr = 7 bytes long. Each STAFF record has the following fields: Name (20 bytes), Ssn (9 bytes), Department (9 bytes), Address (40 bytes), Phone (10 bytes), Salary (8 bytes), Position_code (4 bytes), Email (32 bytes), and Job_description (200 bytes). An additional byte is used as a deletion marker. Suppose that the file is ordered by the key field Ssn and we want to construct a primary index on Ssn.

   a. Calculate the blocking factor (bfr) and the number of file blocks b needed to store the STAFF records. Assume that records are stored unspanned. How much space remains unused per block?

   Solution:
   The record size R = 20 + 9 + 9 + 40 + 10 + 8 + 4 + 32 + 200 + 1 = 333
   To calculate the blocking factor, we get bfr = floor(B / R) = 3 records/block.
   Therefore, the unused space per block is B – (R * bfr) = 25 bytes.

   b. Calculate the index blocking factor bfr_i.

   Solution**:**
   Each index entry has size Ri = size(ssn) + P = 9 + 6 = 15 bytes.
   The index blocking factor is bfri = floor(B / Ri) = 68 entries / block.

   c. Assume that we only have a single-level index. Calculate the number of index entries and the number of index blocks.

   Solution:
   We need one index entry per file block, so to calculate the number of index entries we calculate the number of blocks needed to store all 40,000 file records, which is
   I = ceiling(40,000 / bfr) = 13334 blocks.
   The number of index blocks Bi = ceil(I / bfri) = 197 blocks.

   d. Now suppose that we want to make the index a multilevel index. How many levels are needed and what is the total number of blocks required by the multilevel index?

   Solution:
   The fan-out for the multilevel index is the same as the index blocking factor (bfri) which is 68 (see above).
   The number of 1st level blocks L1 is already calculated in 2b, and so

we have L1=197.
The number of 2nd level blocks is L2 = ceil(L1 / fo) = 3.
The number of 3rd level blocks is ceil(L2 / fo) = 1, and therefore we need 3 levels in total.
The total number of index blocks is L1 + L2 + L3 = 197 + 3 + 1 = 201 blocks.

e. Consider the multilevel index from question (d). What is the number of block accesses needed to search for and retrieve a record from the file given its Ssn value?

Solution:
This is given as the number of index levels + 1, which, according to c) is 3+1=4.