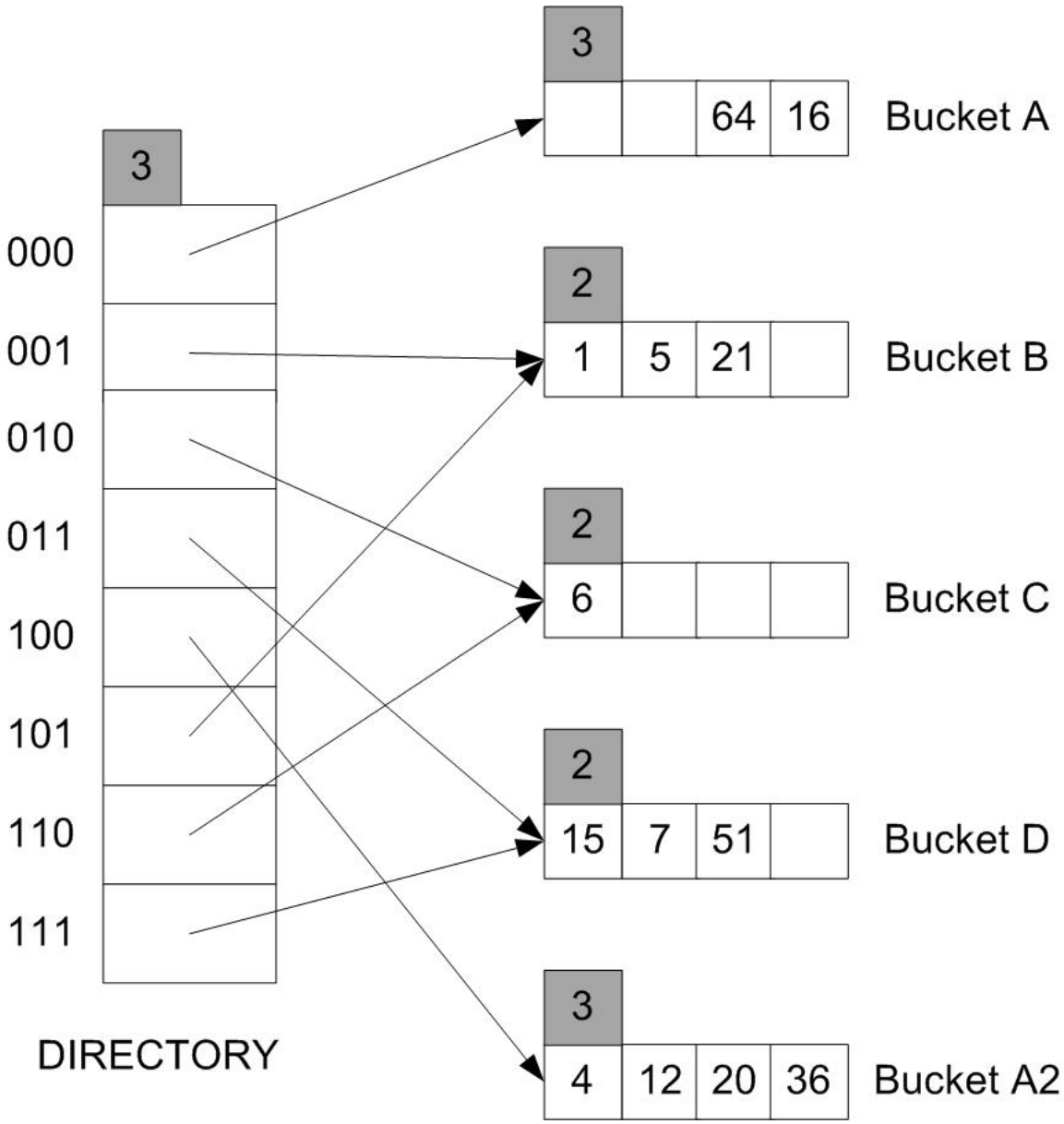
Problem 1: (Hashing)

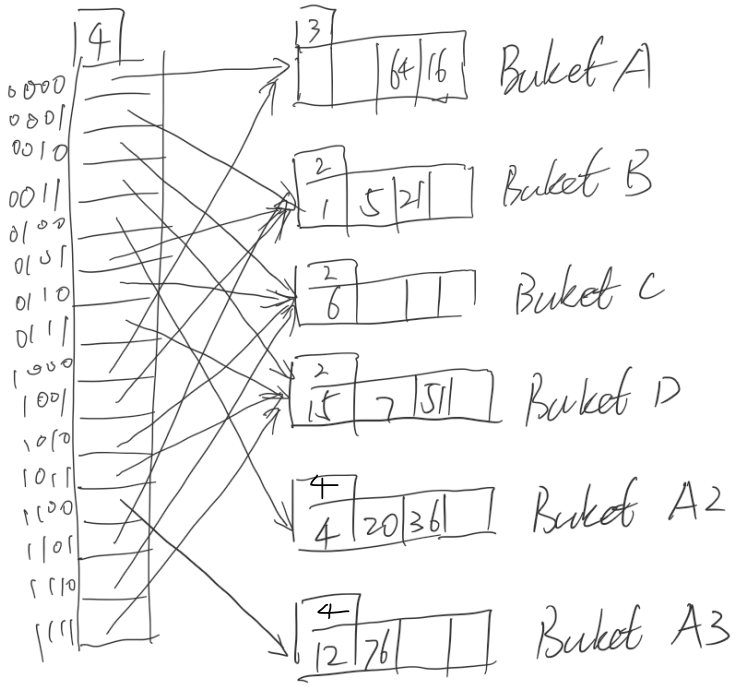
Consider the Extendible Hashing index shown in the figure below. Answer the following questions about this index:



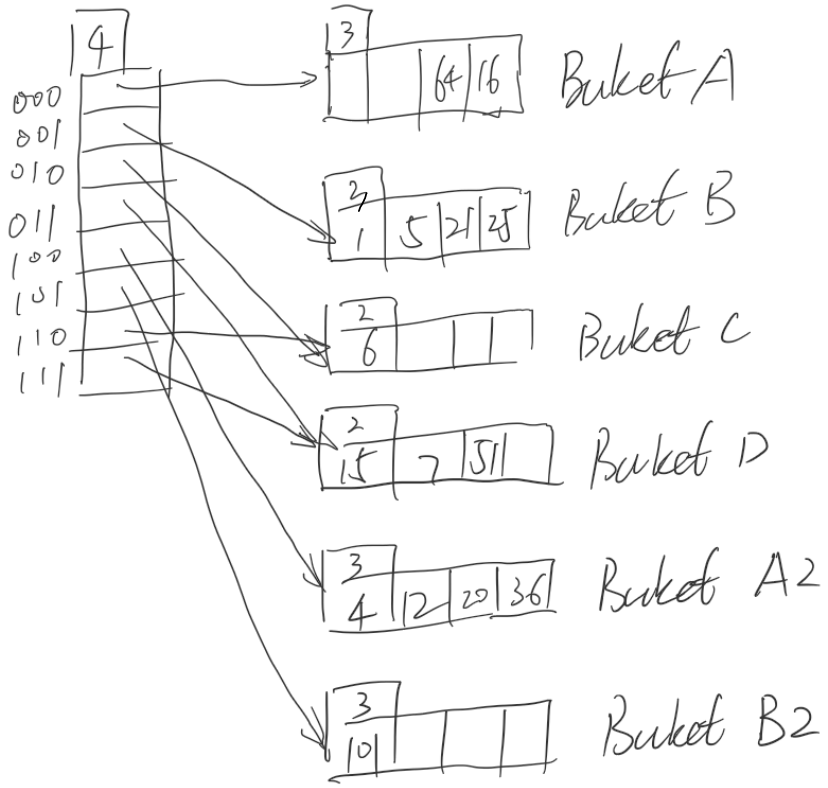
1. Denote by M the number of insert data entries in the index after the occurrence of the last split. What is the *minimum* value of M. Argue your answer.

   M = 1. As we can see from the figure that the last split is cause by an insertion to bucket A. The capacity of a bucket is 4, so M = size(A2) + size(A) – 4 – 1 = 1 (the insertion that triggered the last split is not counted).

2. Show the index after inserting an entry with hash value 76.



3. Show the *original* index after inserting entries with hash values 25 and 101.



Problem 2: (Indexing)

Consider the following relations:
Emp(eid: integer, ename: varchar, sal: integer, age: integer, did: integer)
Dept(did: integer, budget: integer, floor: integer, mgr_eid: integer)

Salaries range from $10,000 to $100,000, ages vary from 20 to 80, each department has about 5 employees on average, there are 10 floors, and budgets vary from $10,000 to $1M. You can assume uniform distribution. For each of the following queries, which of the listed index choices would you choose to speed up the query? Explain briefly (no more than 3 lines).

1. Query 1: Print ename, age, and sal for all employees.
    1. Primary (clustered) hash index on (ename, age, sal) fields of Emp.
    2. Secondary (unclustered) hash index on (ename, age, sal) fields of Emp.
    3. Primary B+ tree index on (ename, age, sal) fields of Emp
    4. Secondary hash index on (eid, did) fields of Emp.
    5. No index.

    I would choose 5.

    Because we simply fetch all the records in Emp relation, which do not need any index at all.

2. Query 2: Find the dids of departments that are on the 10th floor and have a budget of less than $15,000.
    1. Primary index on the floor field of Dept.
    2. Secondary index on the floor field of Dept.
    3. Primary B+ tree index on (floor, budget) fields of Dept.
    4. Secondary B+ tree index on (floor, budget) fields of Dept.
    5. No index.

    I would choose 3.

    Because its faster to use B+ tree index to locate the first department on the 10^{th} floor, and all the records has the same floor number are ordered by budget, which makes it faster to find records within a range of budget on that floor.

Problem 3: Files

Consider a disk with a sector size of 512 bytes, 2,000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec, the disk platters rotate at 5,400 rpm (revolutions per minute). Suppose that a block size of 1,024 bytes is chosen. Suppose that a file F

containing 100,000 records of 100 bytes each is to be stored on such a disk and no record is allowed to span two blocks.

1. How many records fit onto a block?

   ceil (1024/100) = 10 records/block.

2. How many blocks are required to store the entire file F?

   100,000/10 = 10,000 blocks.

3. How many records of 100 bytes each can be stored using this disk?

   Total sector number of this disk:

   50*2000*2*5 = 1,000,000 sectors

   Two sectors make a block, so there are 500,000 blocks in total.

   10 records fit in a block, thus 5,000,000 records can be stored using this disk.

4. What time is required to read the file F sequentially?

   Total number of sectors needed: 10000*2 = 20,000. Tracks number = 20000/50 = 400. And thus the total scanning time of the file:

   400/(5400/60) = 4.44s

5. What time is required to read the file F randomly?

   If we ignore the reading time of each block, then the total random-access time would be:

   10000*10/1000 = 100s.

Problem 4: External Sorting

Answer the following questions for each of these scenarios, assuming that our most general external sorting algorithm is used:

A. A file with 100,000 pages and 30 available buffer pages.
B. A file with 200,000 pages and 50 available buffer pages.
C. A file with 20,000,000 pages and 170 available buffer pages.

Questions:

1. How many runs will you produce in the first pass?
    A. ceil (100000/30) = 3334
    B. ceil (200000/50) = 4000
    C. ceil (20000000/170) = 117648
2. How many passes will it take to sort the file completely?
    A. $1 + \left\lceil \log_{29} \left\lceil \frac{100000}{30} \right\rceil \right\rceil = 4$
    B. $1 + \left\lceil \log_{49} \left\lceil \frac{200000}{50} \right\rceil \right\rceil = 4$
    C. $1 + \left\lceil \log_{169} \left\lceil \frac{20000000}{170} \right\rceil \right\rceil = 4$
3. What is the total I/O cost of sorting the file?
    A. 2*100000*4=800000
    B. 2*200000*4=1600000
    C. 2*20000000*4=160000000
4. How many buffer pages do you need to sort the file completely in just two passes?

$$\log_{B-1} \left\lceil \frac{P}{B} \right\rceil \leq 1 \Rightarrow B \geq \frac{\left(1 + \sqrt{1 + 4P}\right)}{2}$$

    A. $B = \left\lceil \frac{\left(1+\sqrt{1+4\times 1000000}\right)}{2} \right\rceil = 1001$
    B. $B = \left\lceil \frac{\left(1+\sqrt{1+4\times 2000000}\right)}{2} \right\rceil = 1415$
    C. $B = \left\lceil \frac{\left(1+\sqrt{1+4\times 20000000}\right)}{2} \right\rceil = 4473$

Problem 5: Query Evaluation

Consider the join between relations R and S, where the join condition is R.a = S.b. We are given the following information about the two relations. The cost metric is the number of page I/Os, and the cost of writing out the result should be uniformly ignored.

- Relation R contains 10,000 tuples and has 10 tuples per page.
- Relation S contains 2,000 tuples and also has 10 tuples per page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

Questions:

$$b_r = \frac{10000}{10} = 1000$$
$$b_s = \frac{2000}{10} = 200$$
$$B = 52$$

1. What is the cost of joining R and S using a simple nested loops join?

With R as the outer relation:

$$n_r \times b_s + b_r = 2001000$$

With S as the outer relation:

$$n_s \times b_r + b_s = 2000200$$

2. What is the cost of joining R and S using a block nested loops join?

With R as the outer relation:

$$\left\lceil \frac{b_r}{B-2} \right\rceil \times b_s + b_r = 5000$$

With S as the outer relation:

$$\left\lceil \frac{b_s}{B-2} \right\rceil \times b_r + b_s = 4200$$

3. What is the cost of joining R and S using a sort-merge join?

Assume that all tuples for any given values of the join attributes fit in memory:

$$b_r + b_s + 2b_r \times \left(\left\lceil \log_{B-1} \left\lceil \frac{b_r}{B} \right\rceil \right\rceil + 1 \right) + 2b_s \times \left(\left\lceil \log_{B-1} \left\lceil \frac{b_s}{B} \right\rceil \right\rceil + 1 \right) = 6000$$

4. What is the cost of joining R and S using a hash join?

Assume that the distribution of the hashed data is not skewed, then the maximum page number of partitions:

$$\left\lceil \frac{\max(b_r, b_s)}{B-1} \right\rceil = 20$$

Each partition can be fitted into memory, thus not recursive partitioning required. The cost of hash join:

$$3(b_r + b_s) = 3600$$

5. How many tuples will the join of R and S produce, at most, and how many pages would be required to store the result of the join back on disk?

Since the attribute b is the primary key for the relation S, any record in R can match at most one tuple in S. So, the maximum number of tuples is $n_r = 10000$. The required number of page to store the result:

$$\frac{10000}{5} = 2000$$

6. If secondary B+ indexes existed on R.a and S.b, would either provide a cheaper alternative for performing the join (using an index nested loops join) than a block nested loops join? Explain.

   Since relation S has less blocks, use S as outer relation. In worst case, each tuple in S need to perform an index lookup on R. Suppose the height of the B+ is 4, then the cost would be:

   $$b_s + n_s \times (4 + 1) = 10200$$

   So the indexes do not provide a cheaper join operation than block nested loops join.

7. If primary B+ indexes existed on R.a and S.b, would either provide a cheaper alternative for performing the join (using the index nested loops algorithm) than a block nested loops join? Explain.

   Yes, we can perform the merge join on the leaves of B+ trees. Suppose that all leaves of any given value in the two B+ trees could be fitted into memory, then the cost would be $b_r + b_s$+ size of the two B+ trees.