CS 245

Midterm Exam Solution – Winter 2015

This exam is open book and notes. You can use a calculator and your laptop to access course notes and videos (but not to communicate with other people). You have 70 minutes to complete the exam.

Print your name: CS 245 Staff

The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

Signed: CS 245 Staff

Problem	Points	Maximum
1	10	10
2	10	10
3	10	10
4	10	10
5	10	10
6	10	10
Total	60	60

Problem 1 (10 points)

For parts **(a)**, **(b)**, consider a B+ tree of order n (where n is odd) that has a depth $L > 1$.

(a) What is the minimum number of record pointers the tree can contain?

$$2\left(\frac{n+1}{2}\right)^{L-1}$$

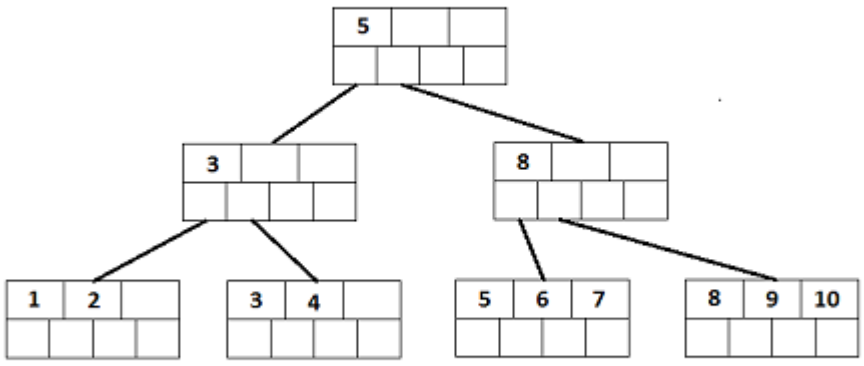
(b) What is the maximum number of record pointers the tree can contain?

$$n(n+1)^{L-1}$$

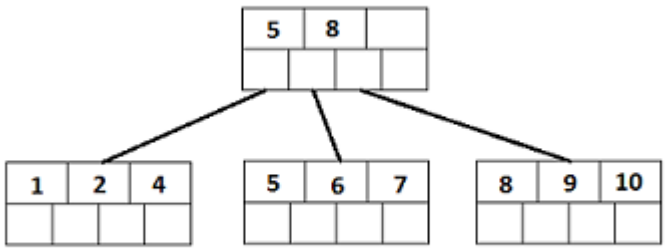
(c) Imagine you have a B+ tree with 3 levels in which each node has exactly 10 keys. There is a record for each key $1, 2, 3, \ldots, N$, where N is the number of records. How many nodes must be examined to find all records with keys in the range $[95, 134]$?

By following the sequence pointers, we only need to look at one node in each of the first two levels. We only need to look at leaf nodes containing a record in this range, of which there are 5 (Node with record # 91-100, 101-110, 111-120, 121-130, 131-140). This means we must read a total of 1 + 1 + 5 = 7 nodes.

(d) Draw the state of the following B+ tree after deleting the record with key 3. Note we are using the textbook's notation for representing nodes.



The resulting B+ tree is shown below:



Problem 2 (10 points)

For this problem consider two relations R and S, with schemas $R(A, B, C)$ and $S(B, C, D)$. The following sub-problems ask you to rewrite given logical queries by manipulating the orders of various selection and projection operations, without changing the final result. We use σ_X as shorthand to denote a selection based on a predicate that involves attributes/conditions X. For example, σ_A represents $\sigma_{A=a}$ (for some value a) and $\sigma_{A \wedge B}$ represents $\sigma_{(A=a) \wedge (B=b)}$ (for some values a, b).

Assume that R and S are sets (not bags). But note that some operators may yield a bag (even if the input was a set), so the final answer may be a bag. All your answers should contain at most 2 join operators.

NOTE: We will give full points if your final answer is correct, but show intermediate steps wherever possible/applicable as you can receive partial points for them if your final answer is incorrect. If you only give an incorrect final answer with no steps, no points will be awarded.

(a) Consider **Expression 1:** $\sigma_B(\sigma_{A \wedge C}(R) \bowtie \sigma_D(S))$.

(a1) Rewrite Expression 1 by pushing the selections as far in as possible (smallest possible join(s)).

$$\sigma_B(\sigma_{A \wedge C}(R) \bowtie \sigma_D(S)) = \sigma_B(\sigma_C(\sigma_A(R) \bowtie \sigma_D(S))) \quad (0.1)$$
$$= \sigma_B(\sigma_{A \wedge C}(R) \bowtie \sigma_{D \wedge C}(S)) \quad (0.2)$$
$$= \sigma_{A \wedge C \wedge B}(R) \bowtie \sigma_{D \wedge C \wedge B}(S) \quad (0.3)$$

$\sigma_{A \wedge B \wedge C}(R) \bowtie \sigma_{B \wedge C \wedge D}(S)$

(a2) Rewrite Expression 1 by pushing the selections as far out as possible (largest possible join(s)).

$\sigma_{A \wedge B \wedge C \wedge D}(R \bowtie S)$

(b) Consider: **Expression 2:** $\sigma_B(\sigma_{A \vee C}(R) \bowtie \sigma_D(S))$.

(b1) Rewrite Expression 2 by pushing the selections as far in as possible (smallest possible join(s)).

$$\sigma_B(\sigma_{A \vee C}(R) \bowtie \sigma_D(S)) = \sigma_B(\sigma_A(R) \bowtie \sigma_D(S)) \cup \sigma_B(\sigma_C(R) \bowtie \sigma_D(S)) \quad (0.4)$$
$$= (\sigma_{A \wedge B}(R) \bowtie \sigma_{D \wedge B}(S)) \cup (\sigma_{C \wedge B}(R) \bowtie \sigma_{D \wedge B}(S)) \quad (0.5)$$
$$= (\sigma_{A \wedge B}(R) \bowtie \sigma_{D \wedge B}(S)) \cup (\sigma_{C \wedge B}(R) \bowtie \sigma_{D \wedge B \wedge C}(S)) \quad (0.6)$$

$$(\sigma_{A \wedge B}(R) \bowtie \sigma_{B \wedge D}(S)) \cup (\sigma_{B \wedge C}(R) \bowtie \sigma_{B \wedge C \wedge D}(S))$$

(b2) Rewrite Expression 2 by pushing the selections as far out as possible (largest possible join(s)).

$$\sigma_{B \wedge (A \vee C) \wedge D}(R \bowtie S)$$

(c) Consider **Expression 3:** $\pi_{A,B}(\sigma_C(R) \bowtie S)$.

Rewrite Expression 3 by pushing the selection and projection as far in as possible (smallest possible join(s)).

$$\pi_{A,B}(\sigma_C(R) \bowtie S) = \pi_{A,B}(\sigma_C(R) \bowtie \sigma_C(S)) \tag{0.7}$$
$$= \pi_{A,B}(\sigma_C(R) \bowtie \pi_{B,C}(\sigma_C(S))) \tag{0.8}$$

(can project out D as it is not part of the join or final projected view)

$$= \pi_{A,B}(\sigma_C(R) \bowtie \pi_B(\sigma_C(S))) \tag{0.9}$$

(can project out D as σ_C ensures join condition over C)

$$= \pi_{A,B}(\sigma_C(R)) \bowtie \pi_B(\sigma_C(S)) \tag{0.10}$$

(can project out C from R as join no longer uses C)

Note that performing the last two steps (projecting out C from R, S) without first pushing σ_C to S is incorrect, as it could result in a potentially different logical result.

$$\pi_{A,B}(\sigma_C(R)) \bowtie \pi_B(\sigma_C(S))$$

Problem 3 (10 points)

Consider a relational DBMS with the following employee relation `E`:

- `E` has attributes `id, name, age, salary`.
- `id` is the primary key attribute (no duplicate ids).
- `E` holds 1024 tuples.
- All records (tuples) in `E` are stored sequentially (in increasing order) based on their `id`.
- Each data block that holds `E` records contains 4 records. (Blocks have been compacted so they are all full.)

We are building a traditional index (not a B+-tree) for this sequential file. Each index block can hold up to 8 (value, pointer) entries (where the pointer can identify either a block or a record). Index entries are sorted by `id`. Index blocks are stored contiguously so we use binary search to look for a key in the index. (We know the address of the first block and the number of index blocks, so we can probe the block in "the middle" at each step.)

(a) Suppose we construct a dense index `INDEX1` on `E.id`, and we search for a record with a given `id`. In the worst case scenario, how many blocks must the system read to retrieve the record? Assume that the a record with the given id exists.

Number of blocks: $8 + 1 = 9$

1024 records fits in 128 `INDEX1` blocks, to perform a binary search and read 1 `INDEX1` block, we need at most $\log_2 128 + 1 = 8$ `INDEX1` reads. We also need 1 record read. Therefore, the total is $8 + 1 = 9$.

In the worst case, `BinSearch`$(2^n) = \log_2 2^n + 1 = n + 1$ reads (need n reads to determine which index block to read and need 1 read to actually read the index block).

(b) To improve access speed, we add an index `INDEX2` on `INDEX1`. What type of index makes sense, sparse or dense?

Type of index: Sparse

(c) After building `INDEX2` we want to find a record with a given `id`. In the worst case scenario, how many blocks must the system read to retrieve the contents of the (existing) record?

To simplify your work for parts (c) through (e), assume you have a function `BinSearch(blocks)` that returns the number of blocks you need to access for a binary search of `blocks` sequential blocks, to find and retrieve a `(id, ptr)` entry that exists in the index. (The result does not include the cost of accessing lower level indexes (if any) or of fetching the actual record.) You can use this function in your answer if you like.

Number of blocks: `BinSearch`$(16) + 2$ [$= 7$]

In the worst case, $\texttt{BinSearch}(2^n) = \log_2 2^n + 1 = n + 1$ reads (need n reads to determine which index block to read and need 1 read to actually read the index block).

Recall from part (a) that we have 128 `INDEX1` blocks. Therefore, there are $\frac{128}{8} = 16$ `INDEX2` blocks. To perform a binary search, we need at most $\texttt{BinSearch}(16) = 5$ `INDEX2` reads + 1 `INDEX1` read + 1 record read

(d) Next let us build a dense *secondary* index on `E.salary` (let us call it `INDEX3`) . Suppose that there are 64 distinct `salary` values, with an equal number of records per salary value. First let us say that we implement the `E.salary` index in a naive way, where duplicate values are stored in the index. That is, if there are x records `salary` = 1000, the index contains x entries of the form (1000, pointer). (Salaries are sorted in the index and the index blocks are stored contiguously.)

Let 1000 be the second largest salary value (i.e., only one salary is larger than 1000). (The search procedure does not know this fact). Say we want to read the content of *all* records with `salary` = 1000. In the worst case scenario, how many blocks need to be fetched?

Number of blocks: $(\texttt{BinSearch}(128) + 1) + 16 = \texttt{BinSearch}(128) + 17$ [= 25]

As 1024 records fits in 128 `INDEX3` blocks, to perform a binary search, we need at most $\texttt{BinSearch}(128) = 8$ `INDEX3` reads. These 8 reads will allow us to find and retrieve the **first** index block containing a (1000, pointer) entry.

Since there are 16 records with `salary` of 1000 (thus 16 (1000, pointer) entries) and each index block can only store 8 (value, pointer) entries, we must read 2 `INDEX3` blocks, not just the first one. Therefore, we need to add 1 read. Lastly, we read 16 records.

(e) Next, suppose we build `E.salary` without duplicate values. For instance, if there are x records with `salary` = 1000, there will be a single entry in the index, pointing to a set of contiguous blocks that store the x record pointers. Salaries are still sorted in the index blocks are stored contiguously. Each pointer block can store 16 pointer entries.

In the worst case scenario, how many blocks need to be fetched to read the content of all records with `salary` = 1000?

Number of blocks: $\texttt{BinSearch}(8) + 17$ [= 21]

64 records (because there are 64 distinct salary values) fits in 8 `INDEX3` blocks, to perform a binary search, we need at most $\texttt{BinSearch}(8) = 4$ `INDEX3` reads + 1 pointer block read (16 pointers to records at 16 per block) + 16 reads for 16 records

Problem 4 (10 points)

Consider two relations $R_1(A, B, C)$, $R_2(B, C, D)$.

We are given the following information:

- $T(R_1) = 100,000; V(R_1, A) = 100; V(R_1, B) = 50; V(R_1, C) = 20.$
- $T(R_2) = 10,000; V(R_2, B) = 20; V(R_2, C) = 50; V(R_2, D) = 10.$
- All attributes are 10 bytes in size.
- We use the "containment of value sets" assumption.
- We assume query values are selected from values in the relations.

Compute the number of tuples and size (in bytes) of the following expressions:

(a) $W = (\sigma_{A=3} R_1) \bowtie (\sigma_{B=5} R_2)$

Let's call $M = \sigma_{A=3} R_1$ and $N = \sigma_{B=5} R_2$.

$$T(W) = \frac{T(M)T(N)}{\max(V(M,B), V(N,B)) \max(V(M,C), V(N,C))} = \frac{\frac{100000}{100} \times \frac{10000}{20}}{50 \times 50} = 200$$

$$S(W) = 4 \times 10 = 40$$

(b) $Y = \pi_{CD}[(\sigma_{B=3 \wedge C=5} R_2)]$

Let's call $P = \sigma_{B=3 \wedge C=5} R_2$. We need to calculate $T(Y) = \max(V(P,C), V(P,D))$ but we know $V(P,C) = 1$, so we need $V(P,D)$. By containment of value sets, we have that $V(P,D) = \min(T(P), V(R_2, D))$. But note that $T(P) = \frac{T(R_2)}{V(R_2,B)V(R_2,C)} = 10$.

$$T(Y) = \min(T(P), V(R_2, D)) = 10$$

$$S(Y) = 2 \times 10 = 20$$

(c) $U = \sigma_{D=1}[(\sigma_{A=3 \wedge B=5} R_1) \bowtie (\sigma_{C=3 \wedge D=5} R_2)]$

Because of $\sigma_{C=3 \wedge D=5}$, we must have that $(\sigma_{A=3 \wedge B=5} R_1) \bowtie (\sigma_{C=3 \wedge D=5} R_2)$ only contains tuples with $D = 5$. Applying $\sigma_{D=1}$ to this relation will yield an empty relation. Therefore,

$$T(U) = 0$$

Problem 5 (10 points)

Consider a hard disk with the following specifications:

- 6000 RPM
- 3.5in in diameter
- 250GB usable capacity
- 100 cylinders, numbered from 1 (innermost) to 100 (outermost).
- Takes time 1+(t/100) milliseconds to move the head t across t cylinders.
- 20% overhead between blocks (gaps).
- Block size is 32 MB.
- transfer rate is 16 GB/sec.
- For this problem 1GB is 10^9 bytes, 1MB is 10^6 bytes.

(a) Based on the specifications, calculate the average rotational delay (in milliseconds) of this disk.

The average rotational delay is half of the maximum rotational delay.

$$\frac{1}{2} \times \frac{60 \text{ s}}{6000} = 5 \text{ ms}$$

(b) Suppose that we have just finished reading a block at track 50, and we next want to read a block at track 10. What is the total read time (time for the desired block to appear in memory)? You can ignore the delays we ignored in class.

We will only consider seek time, rotational delay, and transfer time:

$$\left(1 + \frac{50 - 10}{100}\right) + (5) + \left(\frac{32 \times 10^6}{16 \times 10^9} \times 10^3\right) = 8.4 \text{ ms}$$

(c) Suppose that we have just finished reading a block at track 50, and seven IO requests are currently pending. These request are for blocks at the following tracks, shown in the order in which the requests arrived: `60, 40, 90, 35, 5, 100, 45`. That is, the request for track `60` was the first one that arrived, the request for `40` was the second one, and so on.

Consider the following three disk-scheduling policies:

- FCFS (first-come-first-serve)
- Elevator policy
- SSTF (shortest-seek-time-first)

For each of the policies, list the requests in the order in which they will be serviced. For example, if you write `5, 35, 40, ...` it means that the request for a block at track 5 will be serviced first, followed by the one for track 35, and so on.

Service order for FCFS:

`60, 40, 90, 35, 5, 100, 45`

Service order for Elevator: You can either move up or down first, so there are two possible options. You only need to state one.

`60, 90, 100, 45, 40, 35, 5`

`45, 40, 35, 5, 60, 90, 100`

Service order for SSTF:

`45, 40, 35, 60, 90, 100, 5`

Problem 6 (10 points)

State if the following statements are true or false. Please write TRUE or FALSE in the space provided.

(a) With a linear hash table, when it is necessary to grow the directory, the directory is doubled in size.

FALSE

Explanation: This is true for extensible hashing. For linear hashing, you add one more bucket at a time.

(b) Fixed-format records must always be fixed-size.

FALSE

Explanation: The schema of fixed-format records only specifies number of fields, type of each field, order in record, and meaning of each field. It does not necessarily specify the length of each field. For instance, you can have a fixed-format record with two fields, integer and string, in which the integer specifies the length of the string.

(c) When the index fits in memory and most queries are searching for keys that do not exist in the relation, dense indexes are always better than sparse indexes.

TRUE

Explanation: With the entire dense indexes in memory, the system does not need to access the record on storage at all.

(d) A grid index is only effective when there are two dimensions.

FALSE

Explanation: You can make a multi-dimensional grid. The logic is exactly the same.

(e) Encrypting records in a database makes it harder to build indexes for them.

TRUE

Explanation: Searching for keys is not as simple with encryption.

(f) Stanford ID numbers definitely do not follow a Zipfian distribution.

TRUE

Explanation: The IDs are assigned in order. There is no particular reason why it should follow Zipfian distribution.

(g) A column store is usually superior to the more traditional row store when queries refer to many attributes (columns).

FALSE

Explanation: If the query refers to one attribute, it would be more superior since we only need to look at a small part of the storage. However, for queries referring to multiple attributes, the system still needs to read a big chunk of data from storage.

(h) The expression $\sigma_p(R - S) = \sigma_p(R) - S$ does not hold when R and S are bags.

FALSE

Explanation: Try to convince yourself the expression holds.

(i) In a multi-attribute hash table (as discussed in class), a different hash function can be used for each attribute.

TRUE

Explanation: The only necessary condition is that for each attribute, you need to be consistent with your choice of hash function.

(j) The 5 minute rule is not useful when data resides on an SSD.

We will accept both answers, depending on how you interpret the word *5 minute rule*.

Explanation for TRUE: SSD has a different IO per second rate and a different price. You cannot directly apply the result of the calculation from HDD and main memory directly to SSD. You can, however, use a similar calculation to come up with the appropriate time interval.

Explanation for FALSE: The name 5 minute rule only refers to the technique we use for calculation, not the exact number five. You can use the same technique to come up with the appropriate time interval for SSD.