Assignment (IV)

Due: May 4, 2023

Problem 1 (30 points). Suppose we have a buffer pool with 4 frames.

(i) Assume the current state of the buffer pool is as follows (pin count is omitted for simplicity):

frame	page	dirty	last used
1	4	N	3
2	5	Y	4
3	6	Y	5
4	7	N	6

Consider the following sequence of page operations:

timestamp	operation	page
7	write	3
8	write	5
9	read	6
10	read	7
11	read	8

(a) Assume we adopt **LRU** page replacement policy. Give the state of the buffer pool after the above page operations and compute the buffer pool hit rate.

$$\text{hit rate} = \frac{\text{number of page hits}}{\text{number of page read/write requests}}$$

(b) Assume we adopt **MRU** (Most-Recently-Used) page replacement policy. Give the state of the buffer pool after the above page operations and compute the buffer pool hit rate.

(ii) With the same initial state of the buffer pool as i, consider the following sequence of page operations:

timestamp	operation	page
7	read	3
8	read	4
9	read	5
10	read	6
11	read	7

(a) Assume we adopt **LRU** page replacement policy. Compute the buffer pool hit rate.

(b) Assume we adopt **MRU** page replacement policy. Compute the buffer pool hit rate.

Problem 2 (40 points). Given the following B^+-Tree :

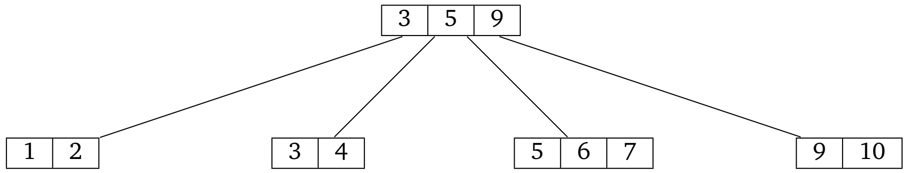

Figure 1: B^+-Tree with max_fanout $n = 4$ and height $h = 2$.

In each of the following questions, you should draw the resulting tree after a key insertion or deletion. Remember the following constraints on B^+-Trees :

- For key-pointer sequence $p_1, k_1, p_2, k_2, \ldots, p_{n-1}, k_{n-1}, p_n$ in an internal node, p_i points to the child node with keys $< k_i$, while p_{i+1} points to the child node with keys $\geqslant k_i$, where $1 \leqslant i < n$.
- The number of **keys** in a **leaf** node should be no less than $\lceil \frac{n-1}{2} \rceil$.
- The number of **children** in an **internal** node should be no less than $\lceil \frac{n}{2} \rceil$.

NOTE: For simplicity, we omit the values and leaf links in the diagram.

(i) Insert 8.

(ii) Start with the tree resulting from i, delete 7.

Problem 3 (30 points). In the lecture, we have presented the *n-ary storage model* (NSM), also known as row storage model. This model is designed to store all attributes for a single tuple contiguously in a page. Consider the table *student* below as an example:

Table 1: Student(id, name, gpa)

id	name	gpa
1	Alice	3.8
2	Bob	3.9
3	Carol	4.0
⋮	⋮	⋮

In NSM, the data page storing *student* may have the following layout.

...	1	Alice	3.8	2	Bob	3.9	3	Carol	4.0	...

The NSM storage model has been widely adopted due to its suitability for *online transaction processing* (OLTP) workloads. This model supports high-speed inserts, updates, and deletes and is good for queries that need the entire tuple. However, NSM is not the optimal choice for queries that involve scanning a large portion or a subset of attributes in the table, which is a common requirement in *online analytical processing* (OLAP) workloads.

In addition to the NSM storage model, there is the option of using the *decomposition storage model* (DSM), also known as the columnar storage model. This model stores values of a single attribute for all tuples contiguously in a page. For example, the data page storing student information in DSM may have the layout depicted below:

...	1	2	3	...	Alice	Bob	Carol	...	3.8	3.9	4.0	...

For simplicity, we assume the values for each attribute are arranged in the order of tuples so that we can identify a tuple with an offset. For example, consider the following query:

```
select gpa from student where name = 'Bob';
```

To process this query, the DBMS first scans through attribute *name* and finds the offset of value *Bob*. Here the offset is 1. Then we get the value in attribute *gpa* with offset 1, which is 3.9.

...	1	2	3	...	Alice	Bob	Carol	...	3.8	3.9	4.0	...
offset	0	1	2		0	1	2		0	1	2	

DSM is ideal for OLAP workloads but is slow for point queries, inserts, updates and deletes.

To help you understand NSM and DSM better, let's conduct a case study on table *student*. Suppose *student* has 3000 tuples that fit into 300 pages and **all attributes have the same width**. Ignore any additional storage overhead (e.g. page headers, tuple headers) and metadata (e.g. table schema, table length) for the table. Additionally, you should make the following assumptions:

- *student* does not have any indices (including for primary key *id*).
- There is no buffer pool in the DBMS. Whenever a page is read/write, it should be fetched from disk first.
- The tuples of *student* can be in any order.

1. Consider the following query:

   ```
   select count(gpa) from student where gpa > 3.7;
   ```

   (a) Suppose the DBMS uses NSM. What is the number of pages that the DBMS will potentially have to read from disk? Write your answer with a short explanation.

   (b) Suppose the DBMS uses DSM. What is the number of pages that the DBMS will potentially have to read from disk? Write your answer with a short explanation.

2. Consider the following query:

   ```
   select name, gpa from student where id = 123;
   ```

   (a) Suppose the DBMS uses NSM. What are the *minimum* and *maximum* numbers of pages that the DBMS will potentially have to read from disk? Write your answer with a short explanation. (Don't forget *id* is primary key!)

   (b) Suppose the DBMS uses DSM. What are the *minimum* and *maximum* numbers of pages that the DBMS will potentially have to read from disk? Write your answer with a short explanation.

Problem 4 (10 points). How long does it take you to finish the assignment? Give a score (1,2,3,4,5) to the difficulty of each problem. List all your collaborators if you have any.