1 Physical Database Design

Consider a relation \( r \) with relation schema \( R(A, B, C) \) and the following additional information:

- \( |r| = 3,000,000 \)
- Relation \( r \) is stored unsorted on disk.
- The domain of attribute \( A \) are integers with range between 1 and 12,000,000
- There are 70\% as many distinct values for attribute \( A \) as there are tuples, i.e., \( |\pi_A(r)| = 0.7 \cdot |r| \). These values are spread uniformly throughout the entire range.
- Size of attribute \( A \): 4 bytes
- Size of a tuple: 32 bytes
- Size of a disk block: 4096 bytes
- Size of a B+-tree pointer: 8 bytes
- The size of a B+-tree node corresponds to the size of a disk block
- The B+-tree fits entirely into main memory (no block transfer cost).

The following two queries are evaluated on relation \( r \):

(a) \( Q_1 : \sigma_{4,000,000 \leq A \land A < 7,000,000}(r) \)
(b) \( Q_2 : \sigma_{A=5,000,000}(r) \)

Tasks:

1. Determine the minimum and maximum path length in a B+-tree on attribute \( A \) and relation \( r \). Explain your approach.

   - Each node in a B+-tree contains \( m - 1 \) search keys and \( m \) pointers. The search key value has the size of a value of attribute \( A \). Then,
     \[
     m \times 8 \text{ bytes} + (m - 1) \times 4 \text{ bytes} = 4096 \text{ bytes}
     \]
     Then,
     \[
     m = \frac{4096 \text{ bytes} + 4 \text{ bytes}}{8 \text{ bytes} + 4 \text{ bytes}} = 341
     \]
   - The internal nodes of a B+-Tree have between \( \left\lceil \frac{m}{2} \right\rceil \) and \( m \) children.
• Thus, the minimum path length occurs when all internal nodes are completely filled and the maximum path length occurs when the internal nodes are half filled.

In relation \( r \), there are 70\% distinct values for attribute \( A \). Thus, there exist \( K = \text{distinctPercentage} \times |r| \) search key values, i.e., \( K = 0.7 \times 3,000,000 \).

**Minimum** path length:

\[
\lceil \log_m(K) \rceil = \lceil \log_{341}(0.7 \times 3,000,000) \rceil = 3
\]

**Maximum** path length:

\[
\lceil \log_{\left\lceil \frac{m}{2} \right\rceil}(K) \rceil = \lceil \log_{\left\lceil \frac{341}{2} \right\rceil}(0.7 \times 3,000,000) \rceil = 3
\]

2. Determine the number of nodes that are traversed in the worst case in the B+-tree on attribute \( A \) in order to fetch all tuples for query \( Q_1 \). Explain your approach.

• In the worst case, every leaf node is only half filled, i.e., it contains \( \lceil \frac{m-1}{2} \rceil = 170 \) search key values.
• Then, there exist \( \left\lfloor \frac{K}{170} \right\rfloor = 12352 \) leaf nodes.
• The predicate of query \( Q_1 \) covers 25\% of the range of attribute \( A \). Tuples must be fetched for 25\% of the search key values and thus, 25\% of the leaf nodes:

\[
12,352 \times 0.25 = 3088 \text{ leaf nodes}
\]

• As the beginning and the end of the predicate’s search range can be partially contained in two leaf nodes, the worst case of traversed leaf nodes is

\[3088 + 2 = 3090 \text{ leaf nodes} \]

• As the maximum path length of the B+-Tree is 3, 2 internal nodes must be traversed. Thus, the total number of traversed nodes in the worst case is

\[
\# \text{ internalNodes} + \# \text{ leafNodes} = 2 + 3090 = 3092
\]

3. Determine the average number of blocks fetched for query \( Q_1 \) using the B+-tree index on attribute \( A \). Explain your approach.

• As the B+-tree fits entirely into main memory, there are no block fetching costs when traversing it.
• The values of attribute \( A \) are uniformly distributed on the range.
• Thus, each tuple has a 25\% chance that its value for attribute \( A \) is within the search range (as the search range covers 25\% of the range of attribute \( A \)).
• Each tuple of the relation is referenced in the B+-tree either by a
direct pointer to the tuple’s block or via a pointer to a bucket (if
more than one tuple refers to the same search key).
• Then, 25% of tuples must be fetched on average and thus, a total of

\[0.25 \times 3,000,000 = 750,000\] blocks

are fetched on average.

4. Determine the number of blocks fetched for query \( Q_1 \) without using the
B+-tree index on attribute \( A \). Explain your approach.

• One disk block contains \( \frac{4096 \text{ bytes}}{32 \text{ bytes}} = 128 \) tuples.
• Then, relation \( r \) is stored in \( \lceil \frac{3,000,000}{128} \rceil = 23,438 \) blocks.
• As the B+-tree index on attribute \( A \) is not used, the entire relation
and thus, all blocks must be fetched.
• A total of 23,438 blocks must be fetched.

5. Determine the average number of blocks fetched for query \( Q_2 \) using the
B+-tree index on attribute \( A \). Explain your approach.

• As the B+-tree fits entirely into main memory, there are no block
fetching costs when traversing it.
• The values of attribute \( A \) are uniformly distributed on the range.
• Thus, each tuple has a 1 to 12,000,000 chance that its value for
attribute \( A \) is 5,000,000.
• Each tuple of the relation is referenced in the B+-tree either by a
direct pointer to the tuple’s block or via a pointer to a bucket (if
more than one tuple refers to the same search key).
• Then, \( \frac{1}{12,000,000} \) of the tuples must be fetched on average and thus, a
total of

\[\frac{3,000,000}{12,000,000} = 0.25\] blocks

are fetched on average.

6. Determine the number of blocks fetched for query \( Q_2 \) without using the
B+-tree index on attribute \( A \). Explain your approach.

• One disk block contains \( \frac{4096 \text{ bytes}}{32 \text{ bytes}} = 128 \) tuples.
• Then, relation \( r \) is stored in \( \lceil \frac{3,000,000}{128} \rceil = 23,438 \) blocks.
• As the B+-tree index on attribute \( A \) is not used, the entire relation
and thus, all blocks must be fetched.
• A total of 23,438 blocks must be fetched.

Note, the number of fetched blocks is independent of the query when the
relation is unsorted and the tuples are retrieved with a sequential scan.
2 Hash Index

Consider the 4 bit hash function \( h(x) = \text{bin}( (x + 2) \mod 15 ) \) and the following extendable hashing scheme with bucket size 3:

Determine the extendable hashing scheme after each of the following insertions:

(a) Insert 61, 51, 75

- \( h(61) = \text{bin}(3) = 0011 \)
- \( h(51) = \text{bin}(8) = 1000 \)
- \( h(75) = \text{bin}(2) = 0010 \)

(b) Insert 69

- \( h(69) = \text{bin}(11) = 1011 \)

(c) Insert 9

- \( h(9) = \text{bin}(11) = 1011 \)
(d) Insert 28

- $h(28) = \text{bin}(0) = 0000$

In the hash function, $\text{bin}(y)$ returns the binary value of $y$ and mod indicates the modulo operation. For instance, the values 12, 14, and 23, which are contained in the hashing scheme, are hashed as follows:

- $h(12) = \text{bin}(14 \mod 15) = \text{bin}(14) = 1110$
- $h(14) = \text{bin}(16 \mod 15) = \text{bin}(1) = 0001$
- $h(23) = \text{bin}(25 \mod 15) = \text{bin}(10) = 1010$